Cluster Analysis

Data Sets: Wholesale customers data and USArrests

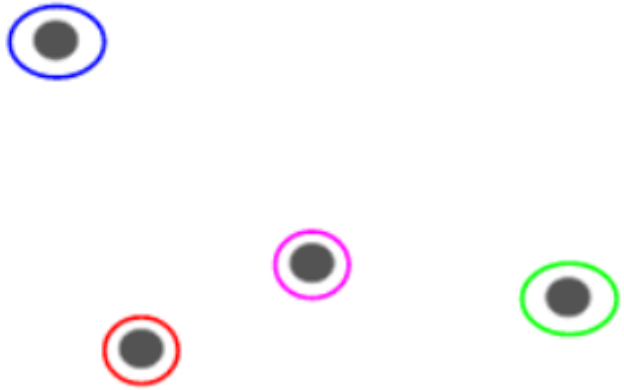
Hierarchical Clustering

What is Hierarchical Clustering?

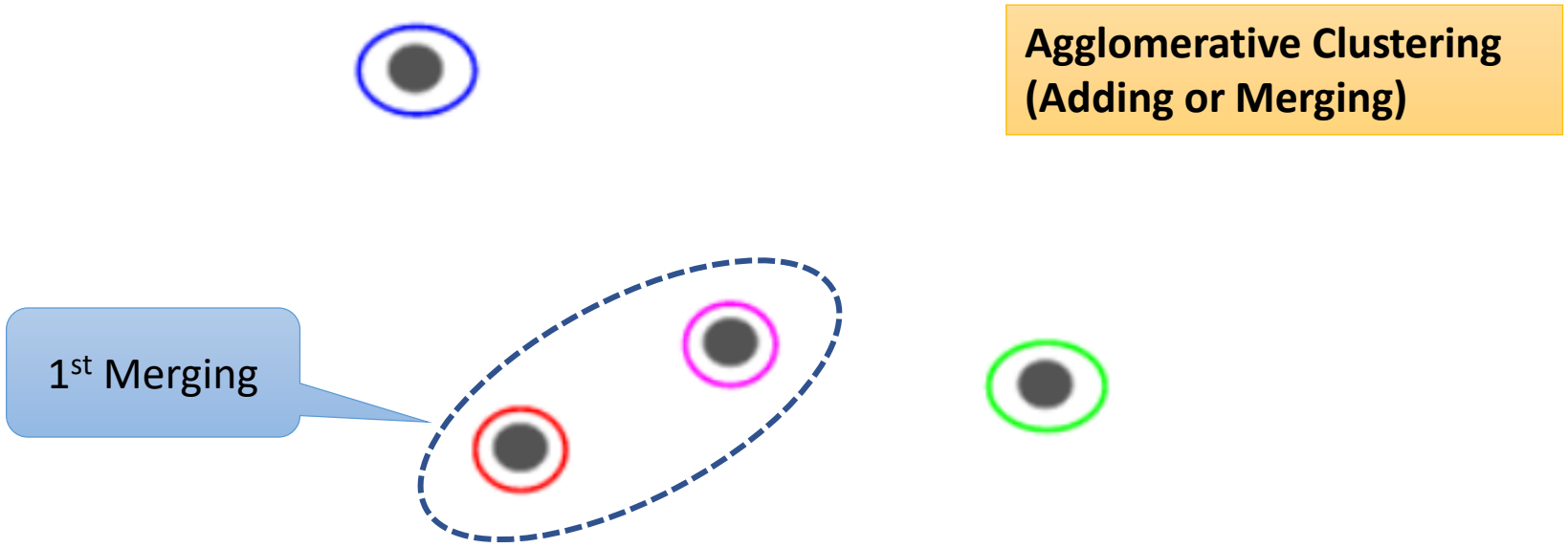
Let's say we have the below points and we want to cluster them into groups:

Each data point as one cluster

We can assign each of these points to a separate cluster:

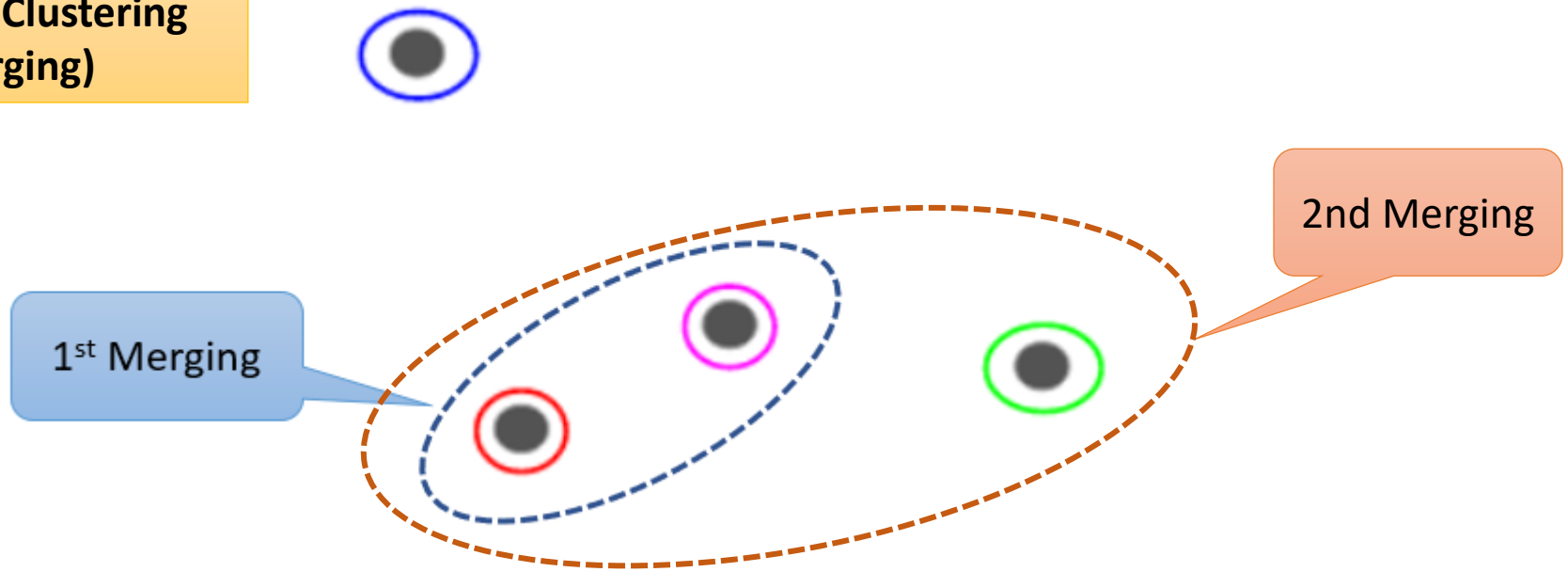


1st merging of two nearest clusters/(data points)



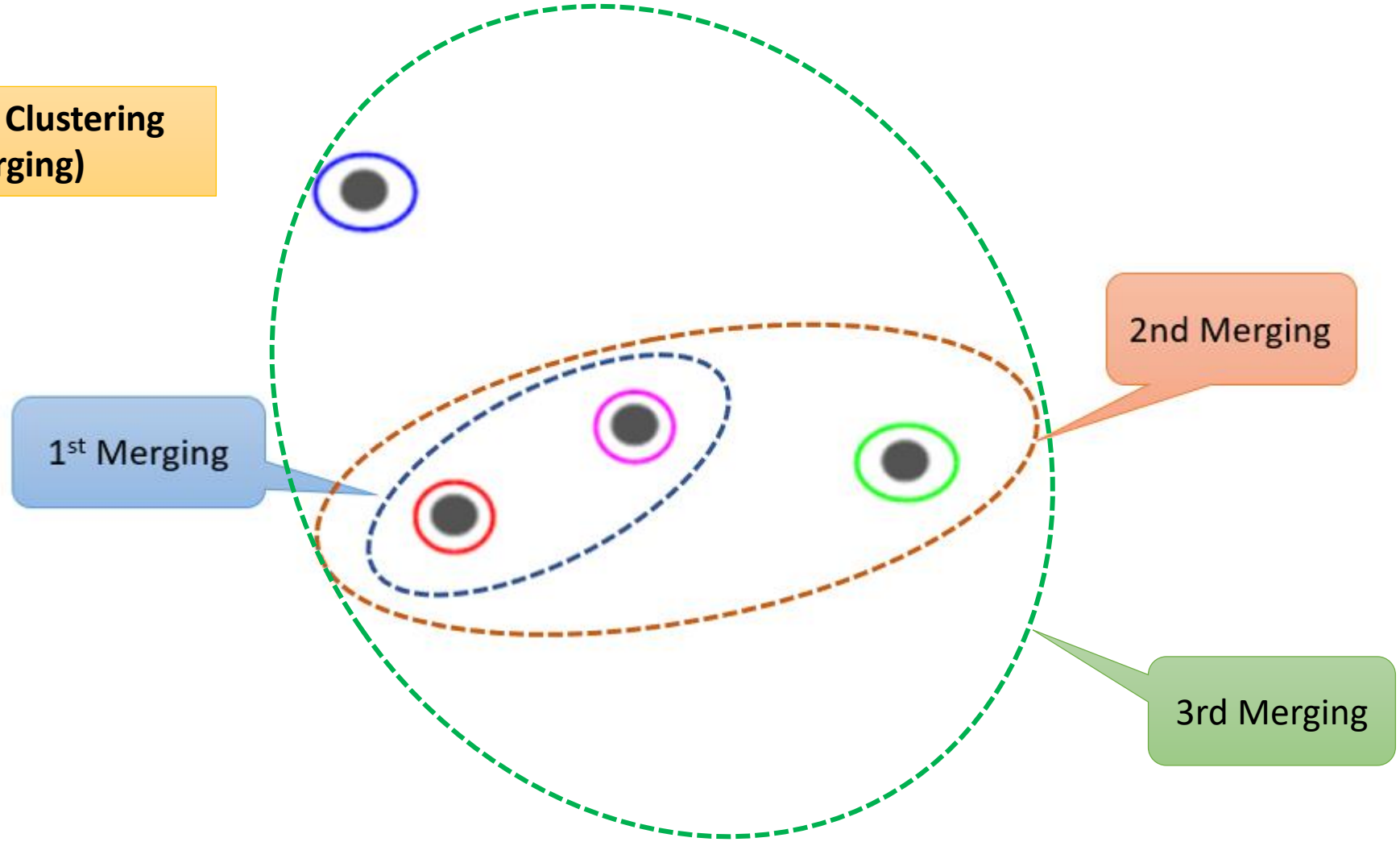
2nd merging of two nearest clusters/(data points)

Agglomerative Clustering (Adding or Merging)



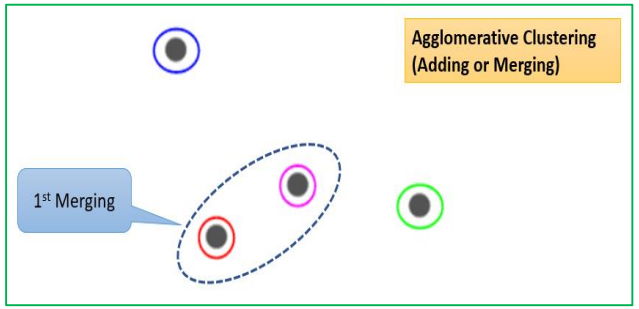
3rd merging of two nearest clusters/(data points)

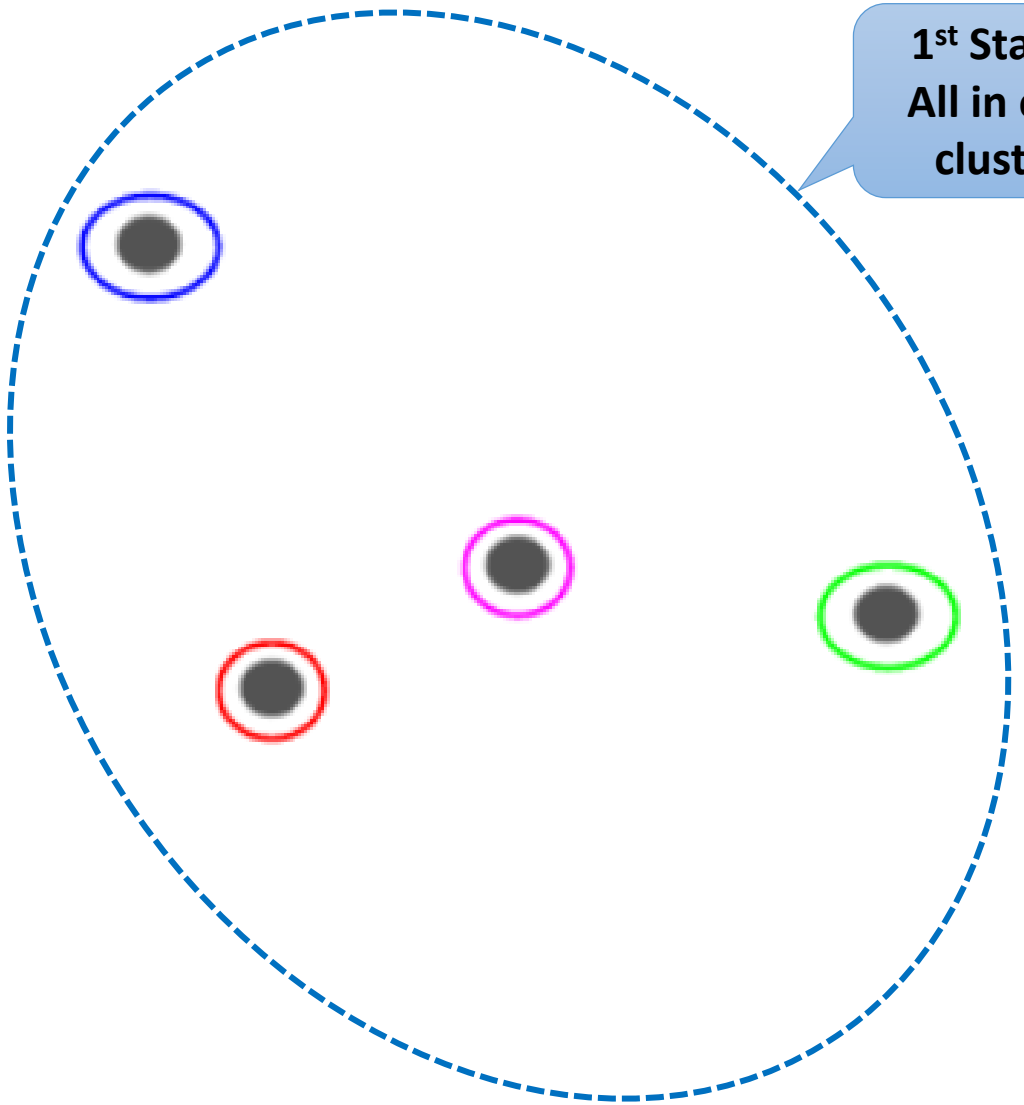
Agglomerative Clustering (Adding or Merging)

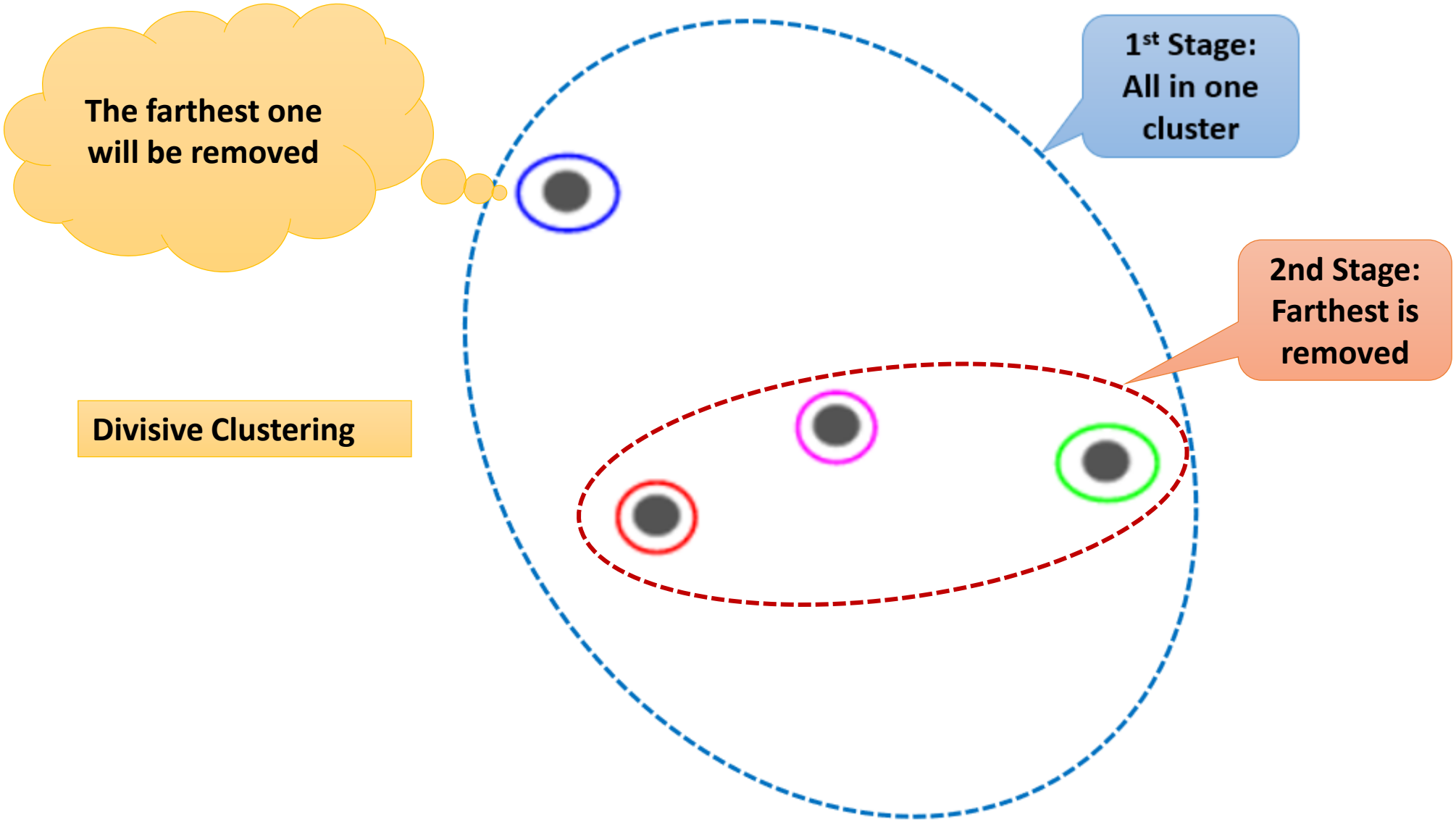


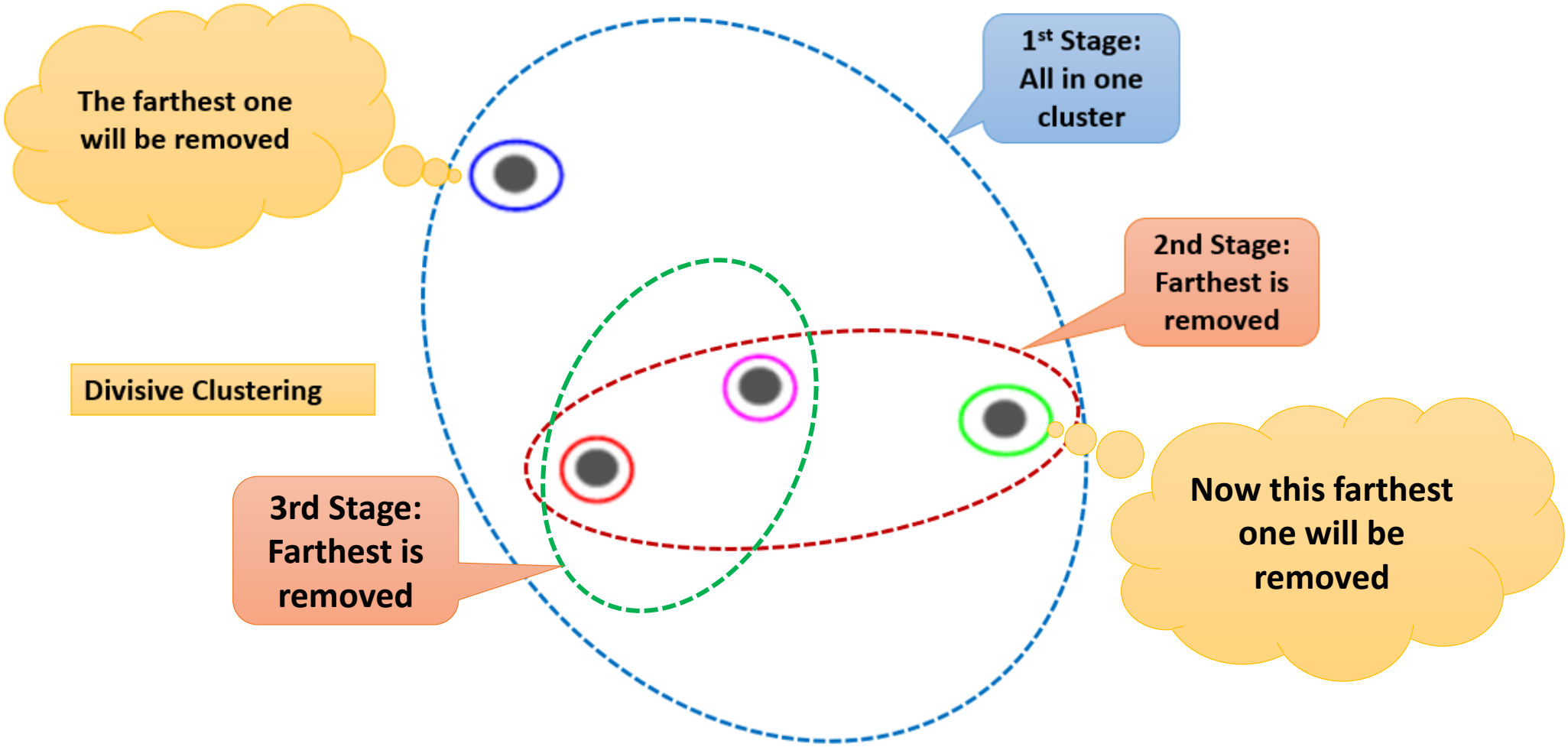
Divisive Hierarchical Clustering

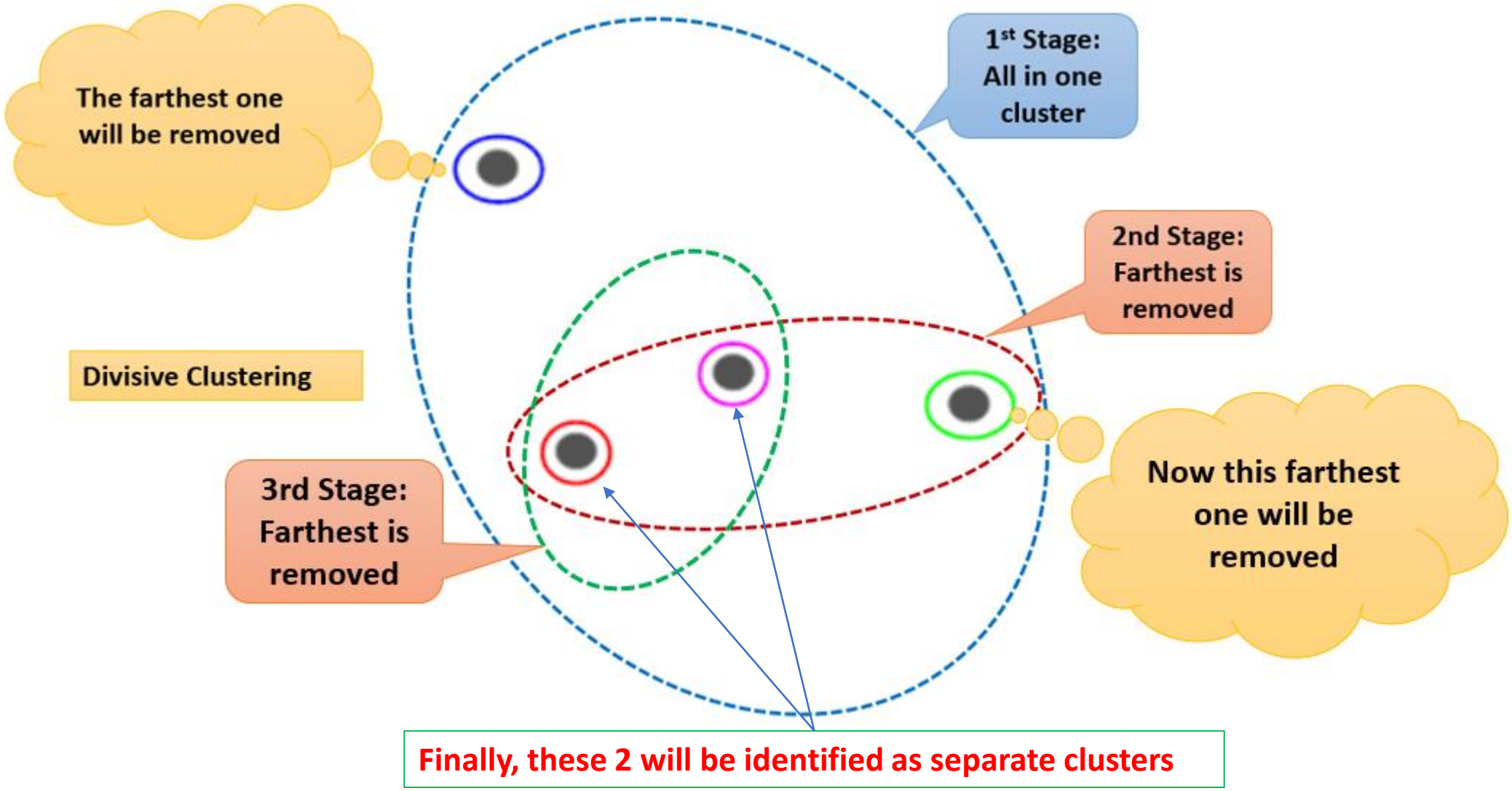
Divisive Clustering (All in one cluster)






The farthest one will be removed
1st Stage: All in one cluster
2nd Stage: Farthest is removed
Divisive Clustering


The farthest one will be removed
1st Stage: All in one cluster
2nd Stage: Farthest is removed
Divisive Clustering
3rd Stage: Farthest is removed
Now this farthest one will be removed


The farthest one will be removed
1st Stage: All in one cluster
2nd Stage: Farthest is removed
Divisive Clustering
3rd Stage: Farthest is removed
Now this farthest one will be removed
Finally, these 2 will be identified as separate clusters

Agglomerative Clustering (Adding or Merging)

ID	Marks
1	10
2	7
3	28
4	20
5	35

$$dist = \sqrt{(x_i - y_i)^2}$$

Stage 1: Five clusters

ID	1	2	3	4	5
1	0	3	18	10	25
2	3	0	21	13	28
3	18	21	0	8	7
4	10	13	8	0	15
5	25	28	7	15	0

As 1 & 2 have smallest distance, so merging them

ID	Marks
1	10
2	7
3	28
4	20
5	35

$$dist = \sqrt{(x_i - y_i)^2}$$

Stage 1: Five clusters

ID	1	2	3	4	5
1	0	3	18	10	25
2	3	0	21	13	28
3	18	21	0	8	7
4	10	13	8	0	15
5	25	28	7	15	0

As 1 & 2 have smallest distance, so merging them

ID	Marks	
1-2	10	max of 10 & 7
3	28	
4	20	
5	35	

Stage 2: Four clusters

ID	1-2	3	4	5
1-2	0	18	10	25
3	18	0	8	7
4	10	8	0	15
5	25	7	15	0

As 3 & 5 have smallest distance, so merging them

ID	Marks	
1-2	10	max of 10 & 7
3	28	
4	20	
5	35	

Stage 2: Four clusters

ID	1-2	3	4	5
1-2	0	18	10	25
3	18	0	8	7
4	10	8	0	15
5	25	**7**	15	0

As 3 & 5 have smallest distance, so merging them

ID	Marks	
1-2	10	
3-5	35	max of 28 & 35
4	20	

Stage 3: Three clusters

ID	1-2	3-5	4
1-2	0	15	10
3-5	15	0	15
4	**10**	15	0

As 1-2 & 4 have smallest distance, so merging them

ID	Marks	
1-2	10	
3-5	35	max of 28 & 35
4	20	

Stage 3: Three clusters

ID	1-2	3-5	4
1-2	0	15	10
3-5	15	0	15
4	10	15	0

As 1-2 & 4 have smallest distance, so merging them

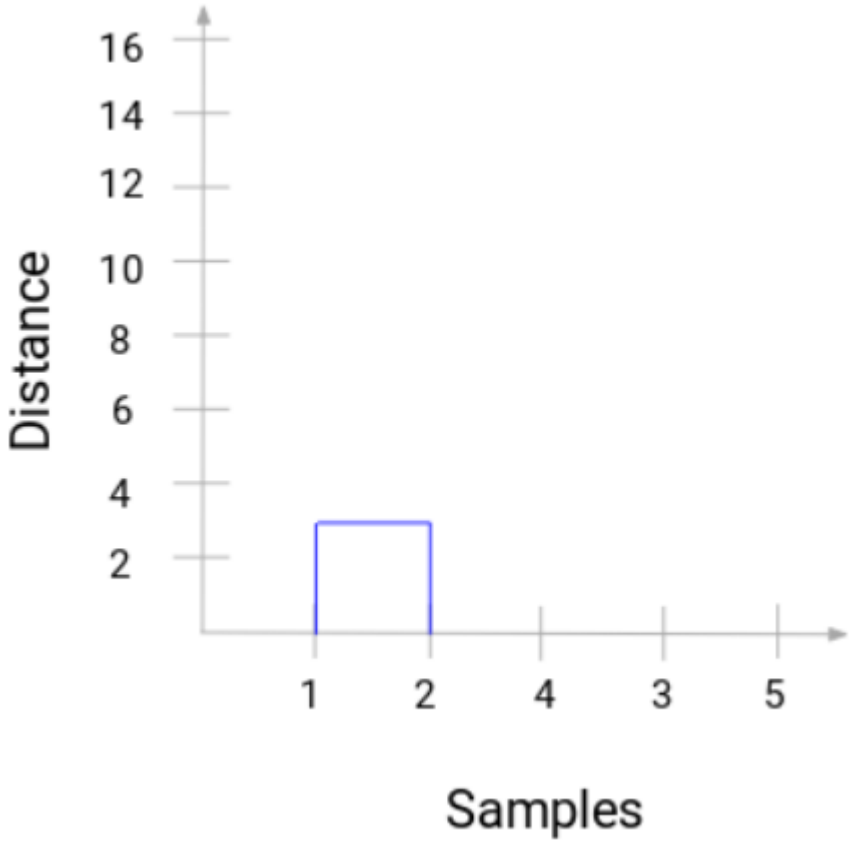
ID	Marks	
1-2-4	20	max of 10 & 20
3-5	35	max of 28 & 35

Stage 4: Two clusters

ID	1-2-4	3-5
1-2-4	0	15
3-5	15	0

As only 2 clusters remained, we merge them

Dendrogram



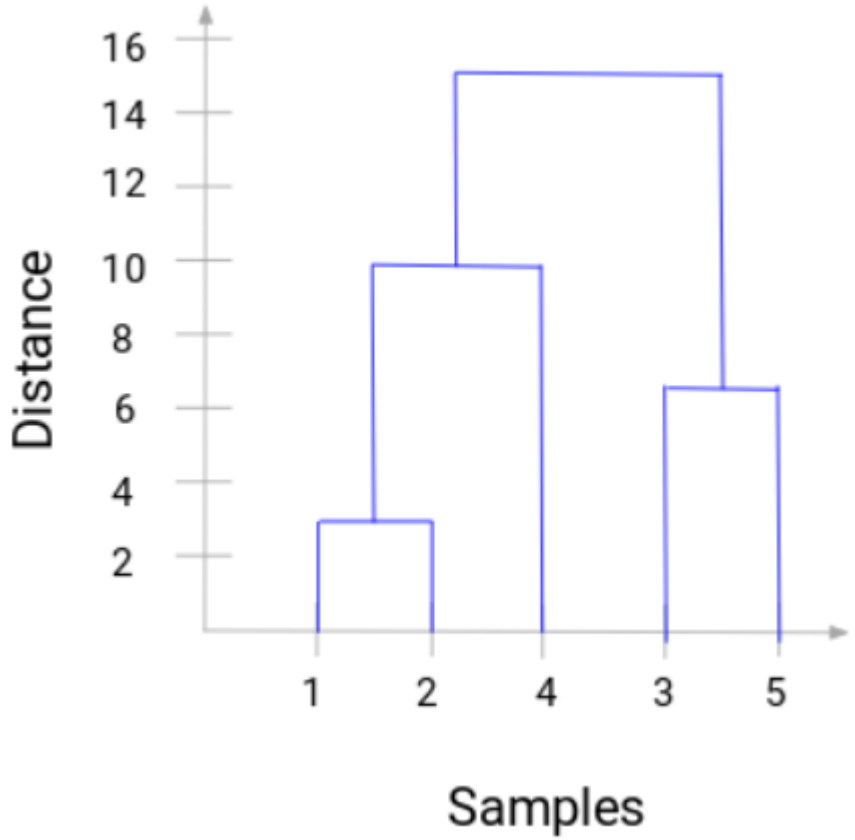
ID	Marks
1	10
2	7
3	28
4	20
5	35

$$dist = \sqrt{(x_i - y_i)^2}$$

Stage 1: Five clusters

ID	1	2	3	4	5
1	0	3	18	10	25
2	3	0	21	13	28
3	18	21	0	8	7
4	10	13	8	0	15
5	25	28	7	15	0

As 1 & 2 have smallest distance, so merging them

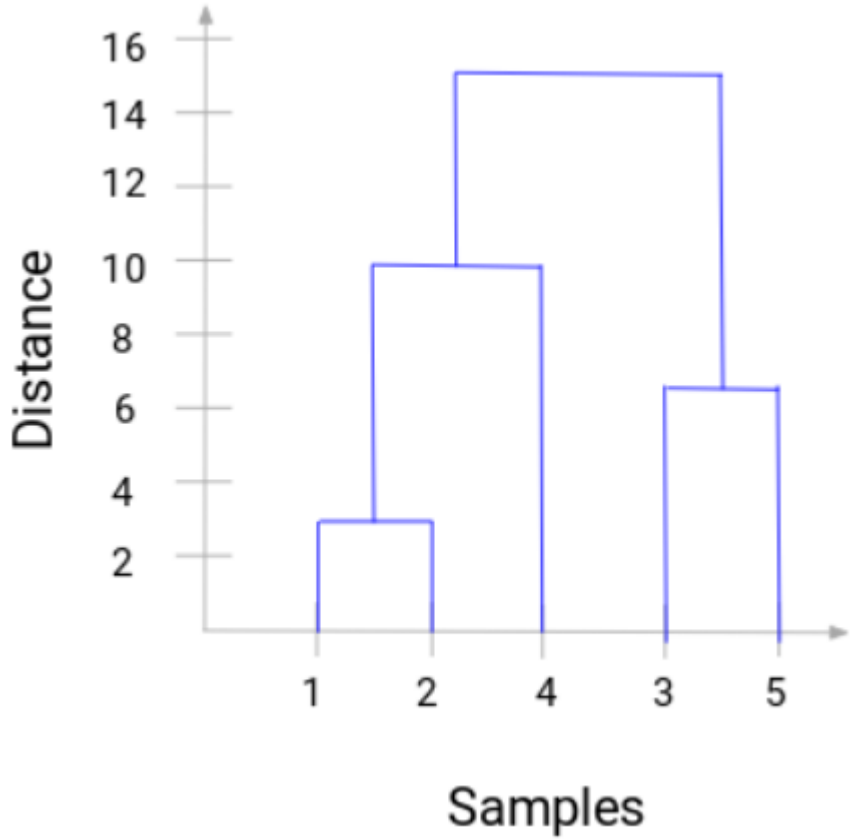


ID	Marks
1-2	10
3	28
4	20
5	35

Stage 2: Four clusters

ID	1-2	3	4	5
1-2	0	18	10	25
3	18	0	8	7
4	10	8	0	15
5	25	7	15	0

As 3 & 5 have smallest distance, so merging them

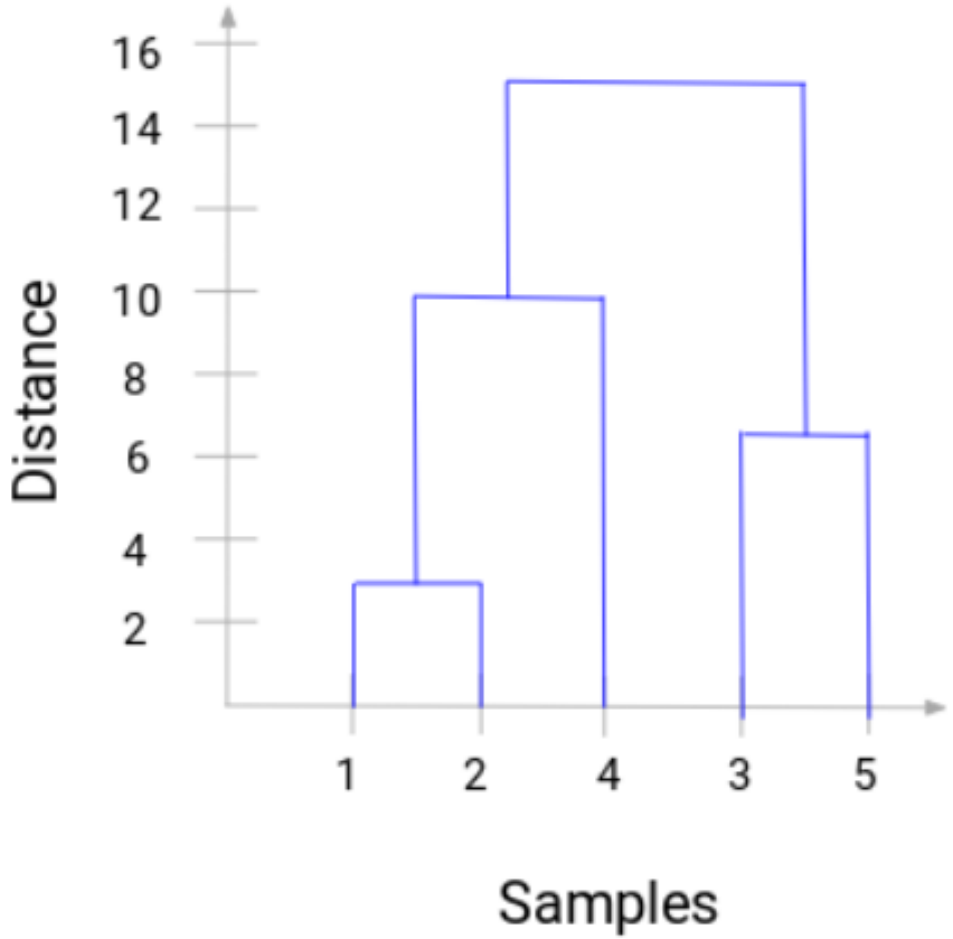


ID	Marks	
1-2	10	
3-5	35	max of 28 & 35
4	20	

Stage 3: Three clusters

ID	1-2	3-5	4
1-2	0	15	10
3-5	15	0	15
4	**10**	15	0

As 1-2 & 4 have smallest distance, so merging them



ID	Marks	
1-2-4	20	max of 10 & 20
3-5	35	max of 28 & 35

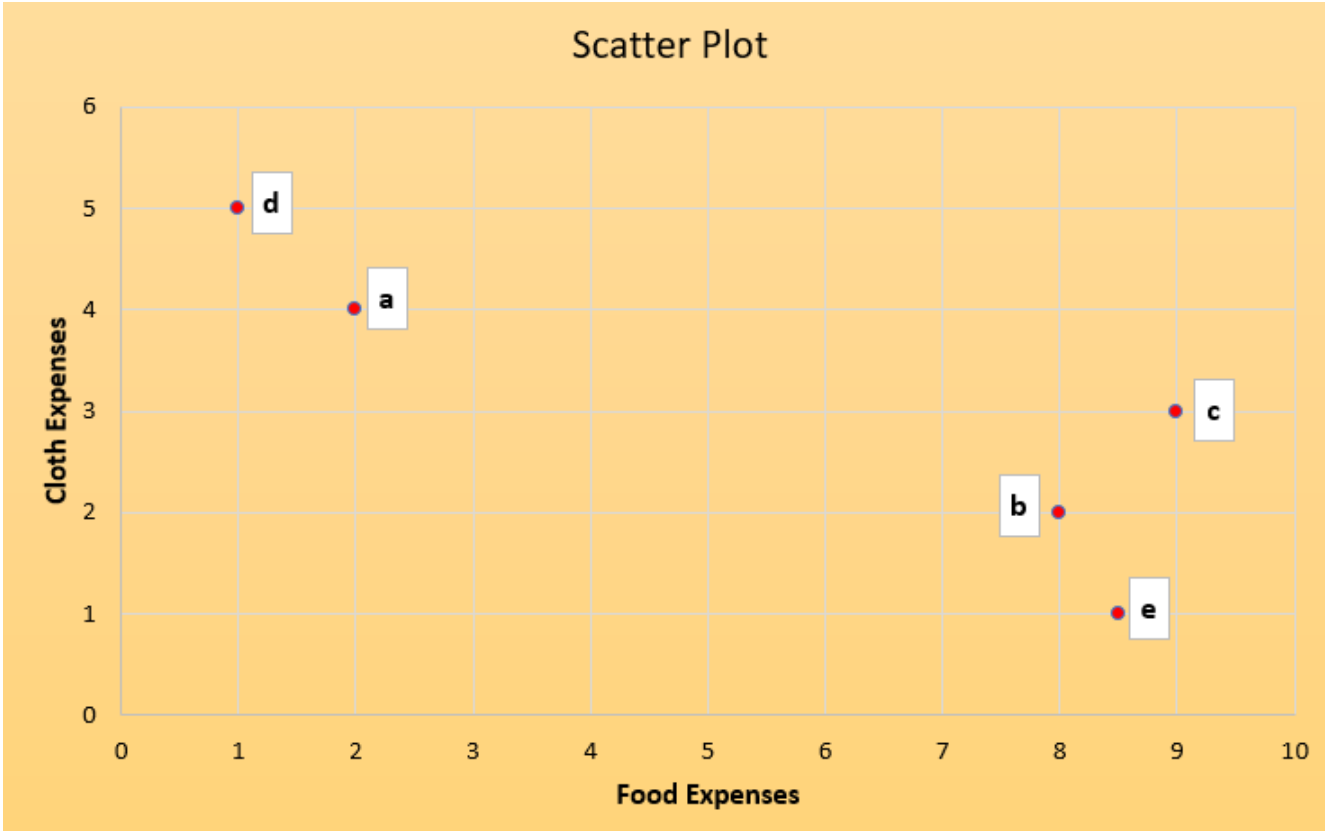
Stage 4: Two clusters

ID	1-2-4	3-5
1-2-4	0	15
3-5	15	0

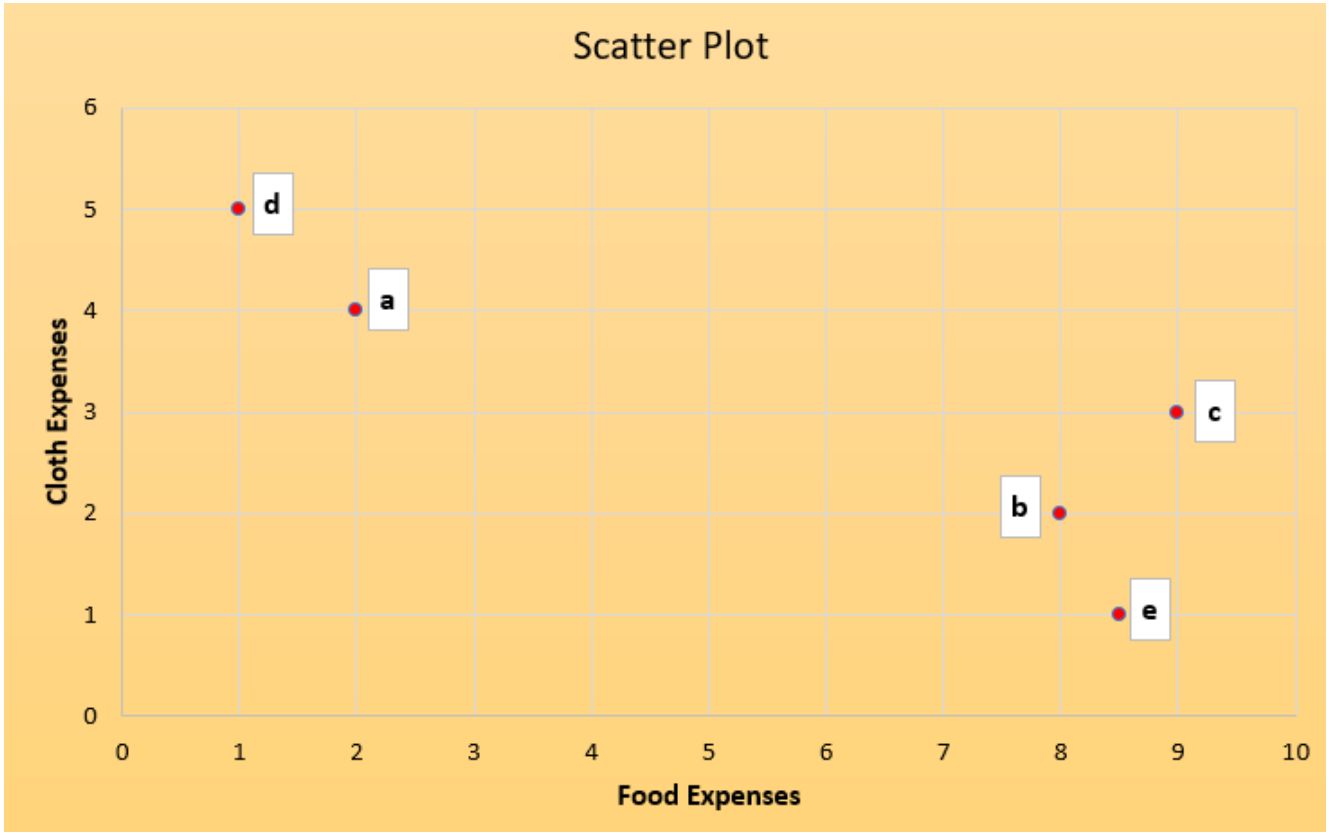
As only 2 clusters remained, we merge them

Data with X and Y: Single Linkage (Min)

	Food Exp	Cloth Exp
	X	Y
a	2	4
b	8	2
c	9	3
d	1	5
e	8.5	1



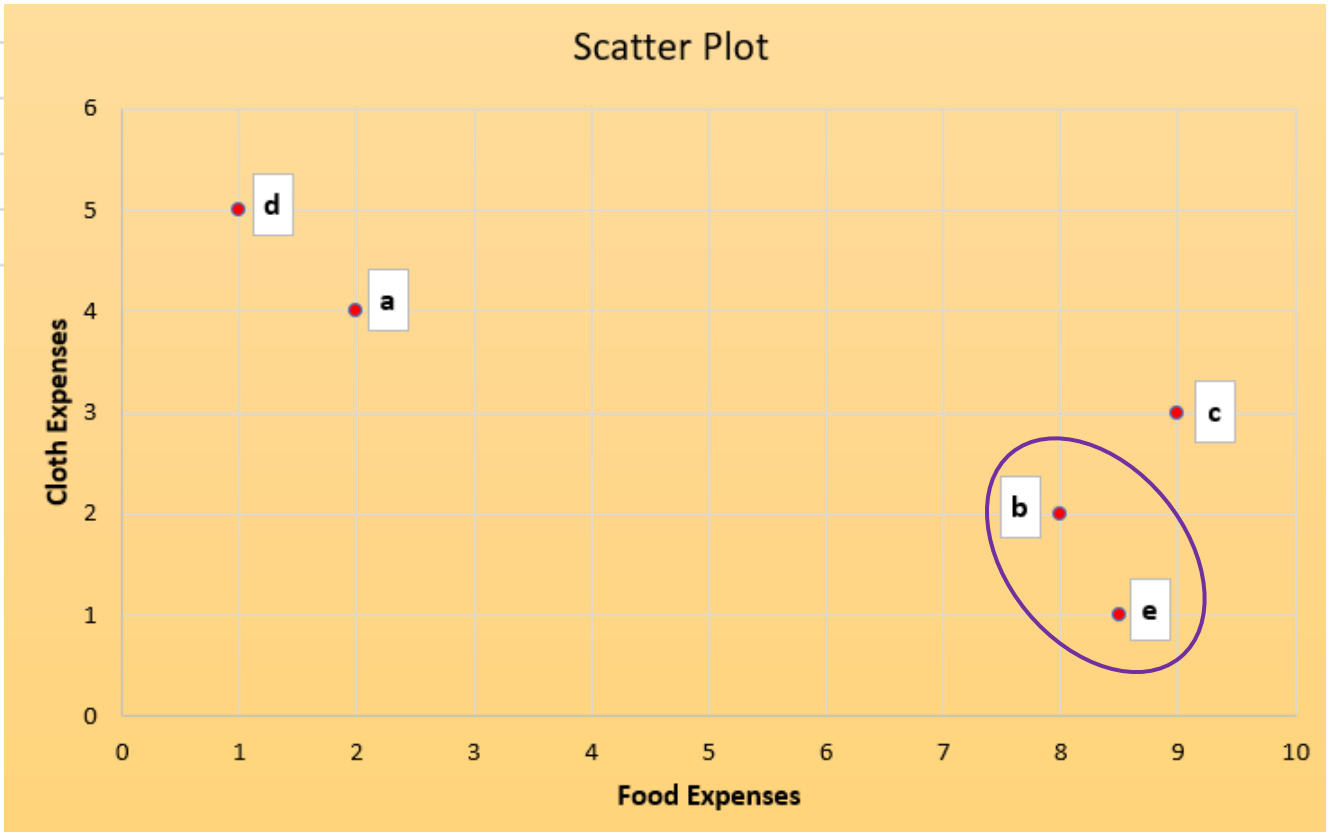
Give a try….make agglomerative process intuitively!



Stage 1: b and e can be clubbed

	a	b	c	d	e
a	0.00	6.32	7.07	1.41	7.16
b		0.00	1.41	7.62	1.12
c			0.00	8.25	2.06
d				0.00	8.50
e					0.00

Now ***b*** and ***e*** are in one cluster. How their values will be treated for finding DIST vis-à-vis another point/s?



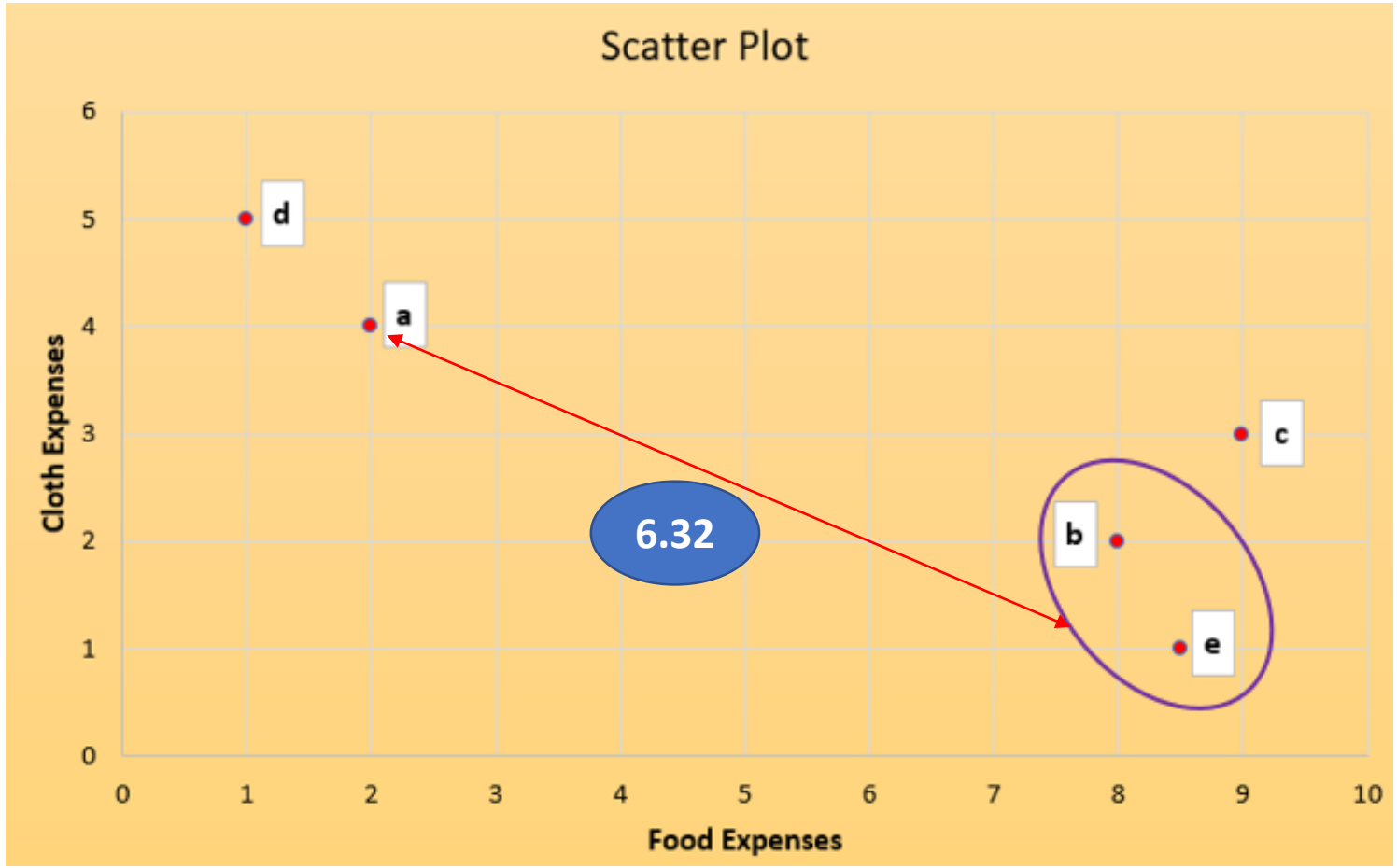
Distance between b, e and a

	a	b	c	d	e
a	0.00	6.32	7.07	1.41	7.16
b		0.00	1.41	7.62	1.12
c			0.00	8.25	2.06
d				0.00	8.50
e					0.00

$$D(be, a) = min\{D(b, a), D(e, a)\}$$

$$D(be, a) = min\{6.32, 7.16\}$$

$$D(be, a) = \mathbf{6.32}$$



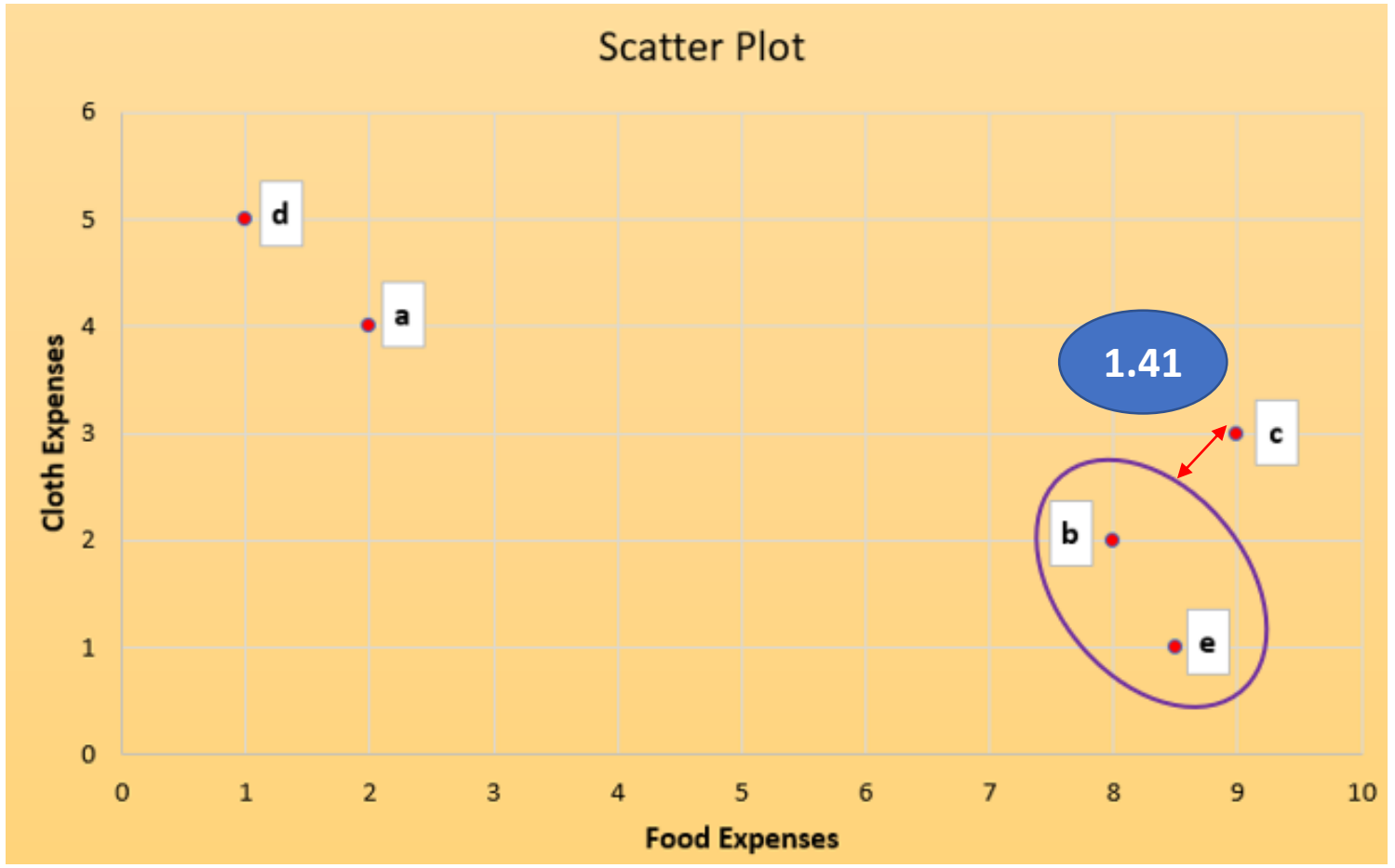
Distance between b, e and c

	a	b	c	d	e
a	0.00	6.32	7.07	1.41	7.16
b		0.00	1.41	7.62	1.12
c			0.00	8.25	2.06
d				0.00	8.50
e					0.00

$$D(be, c) = min\{D(b, c), D(e, c)\}$$

$$D(be, c) = min\{1.41, 2.06\}$$

$$D(be, c) = \mathbf{1.41}$$



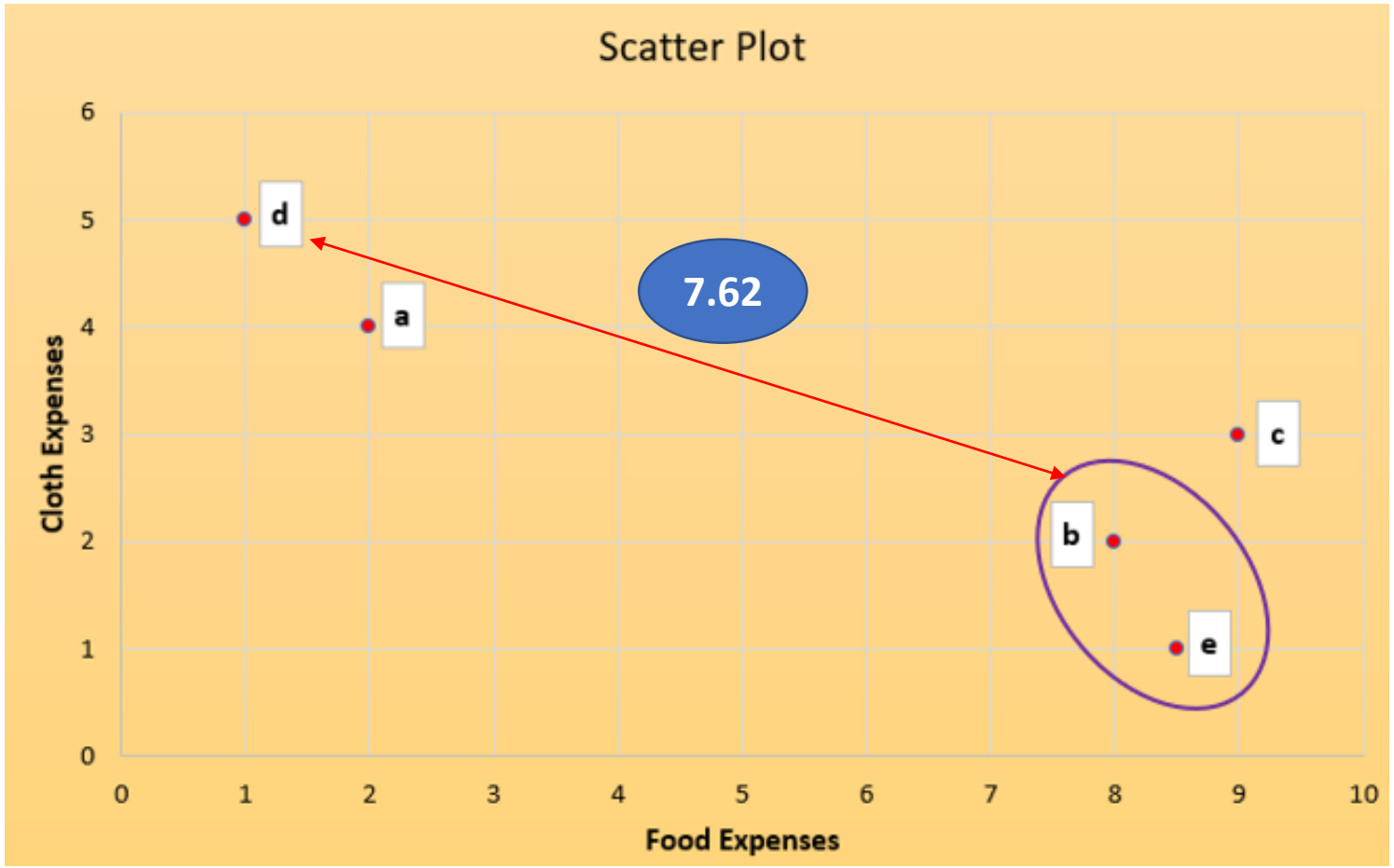
Distance between b, e and d

	a	b	c	d	e
a	0.00	6.32	7.07	1.41	7.16
b		0.00	1.41	7.62	1.12
c			0.00	8.25	2.06
d				0.00	8.50
e					0.00

$$D(be,d) = min\{D(b,d), D(e,d)\}$$

$$D(be,d) = min\{7.62, 8.50\}$$

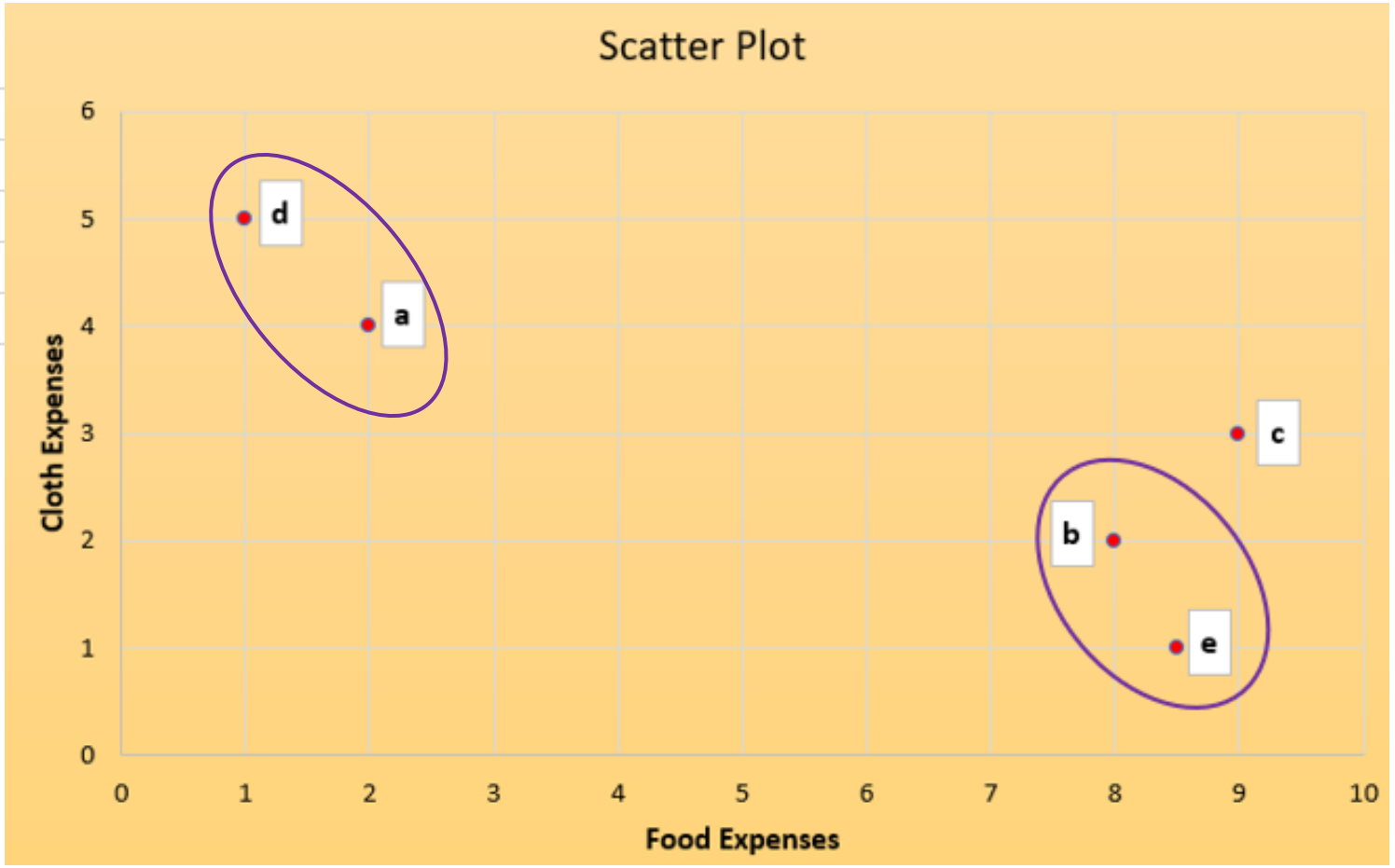
$$D(be,d) = \mathbf{7.62}$$



Stage 2: d and a can be clubbed

	b e	a	c	d
b e	**0**	**6.32**	**1.41**	**7.62**
a		**0**	**7.07**	**1.41**
c			**0**	**8.25**
d				**0**

Distance between clusters (b, e) and (a, d)?



Distance between clusters (b, e) and (a, d)?

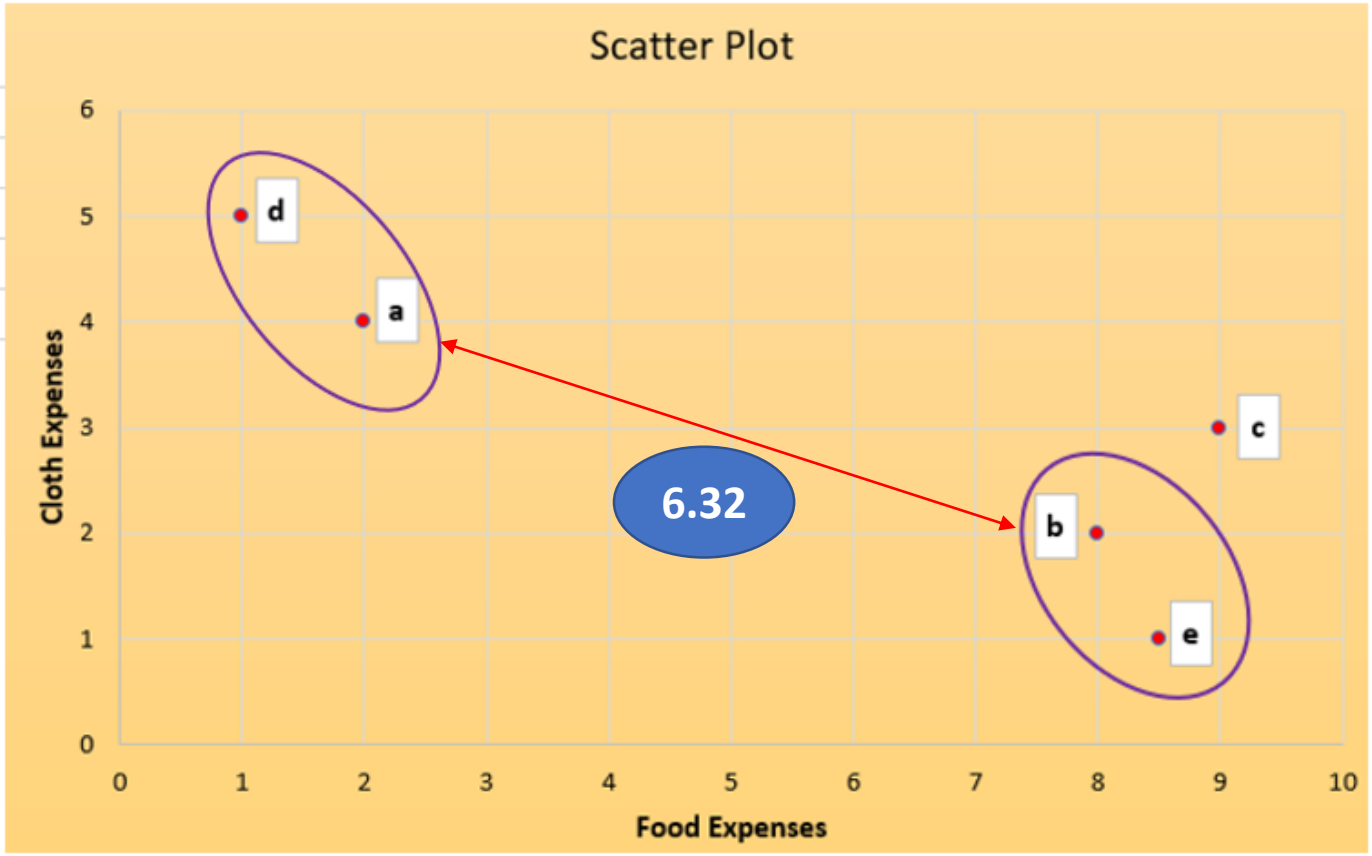
	b e	a	c	d
b e	0	6.32	1.41	7.62
a		0	7.07	1.41
c			0	8.25
d				0

$$D(be, ad) = min\{D(be, a), D(be, d)\}$$

$$D(be, ad) = min\{6.32, 7.62\}$$

$$D(be, ad) = \mathbf{6.32}$$

	b e	a d	c
b e	0	6.32	1.41
a d		0	7.07
c			0



Distance between clusters (a, d) and (c)?

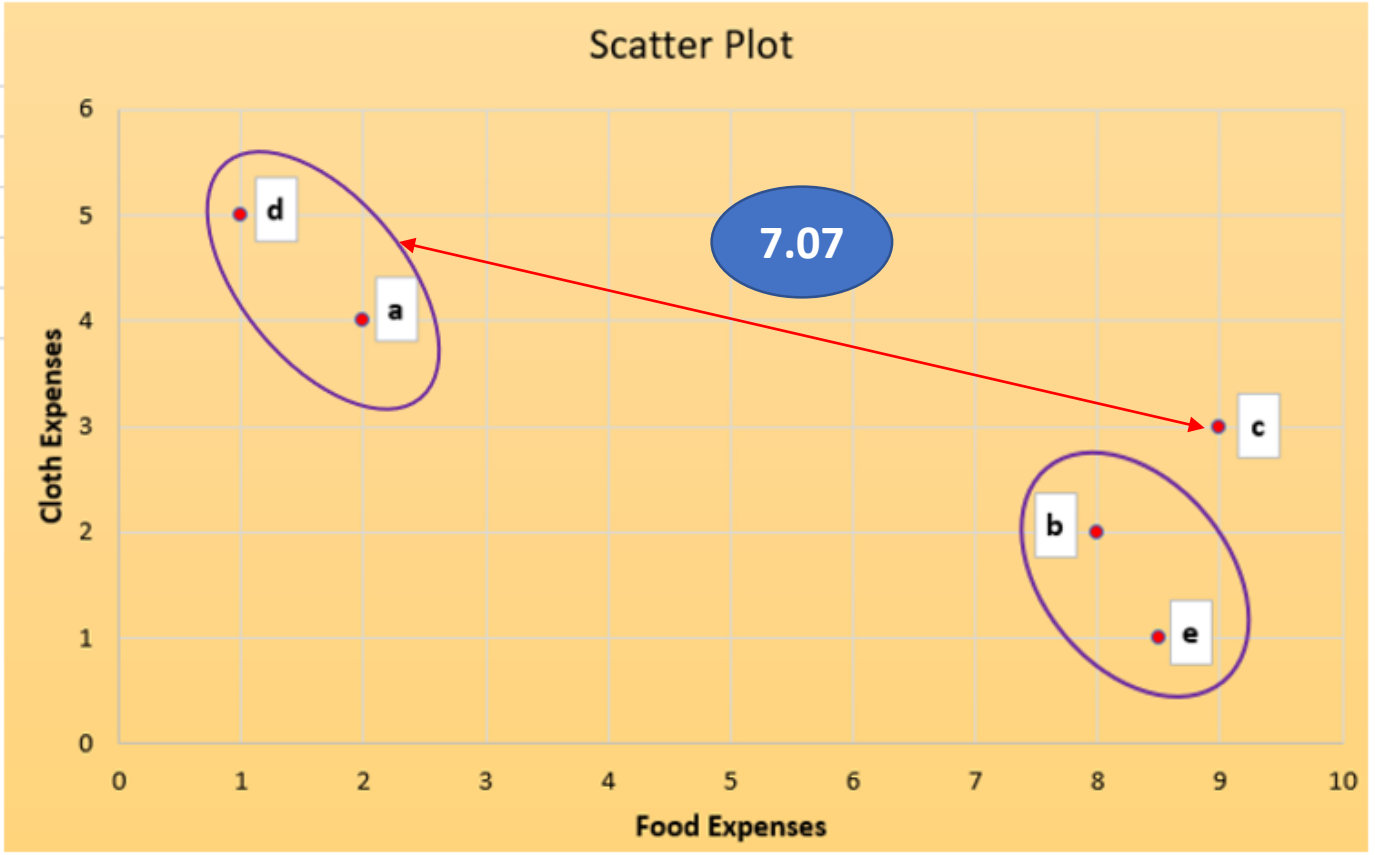
	a	b	c	d	e
a	0.00	6.32	7.07	1.41	7.16
b		0.00	1.41	7.62	1.12
c			0.00	8.25	2.06
d				0.00	8.50
e					0.00

	b e	a d	c
b e	0	6.32	1.41
a d		0	7.07
c			0

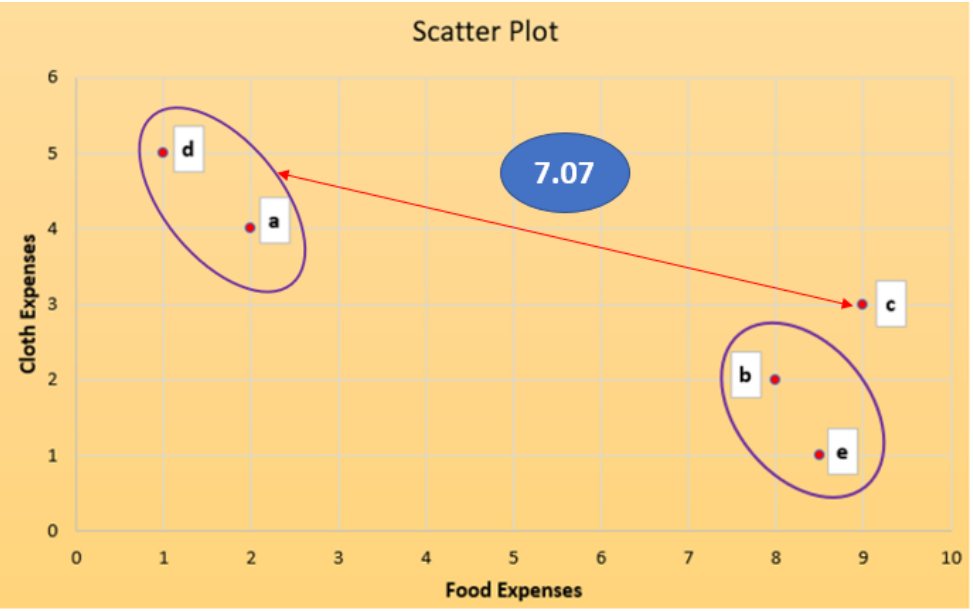
$$D(ad, c) = min\{D(a, c), D(d, c)\}$$

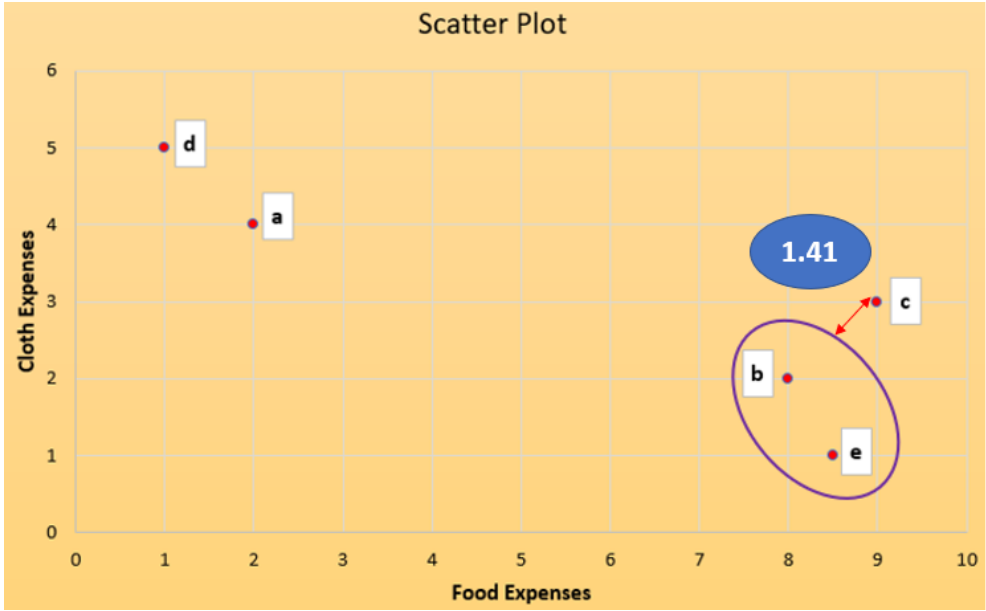
$$D(ad, c) = min\{7.07, 8.25\}$$

$$D(ad, c) = \mathbf{7.07}$$



Where c should go?





Stage 3: Cluster b e must envelop c

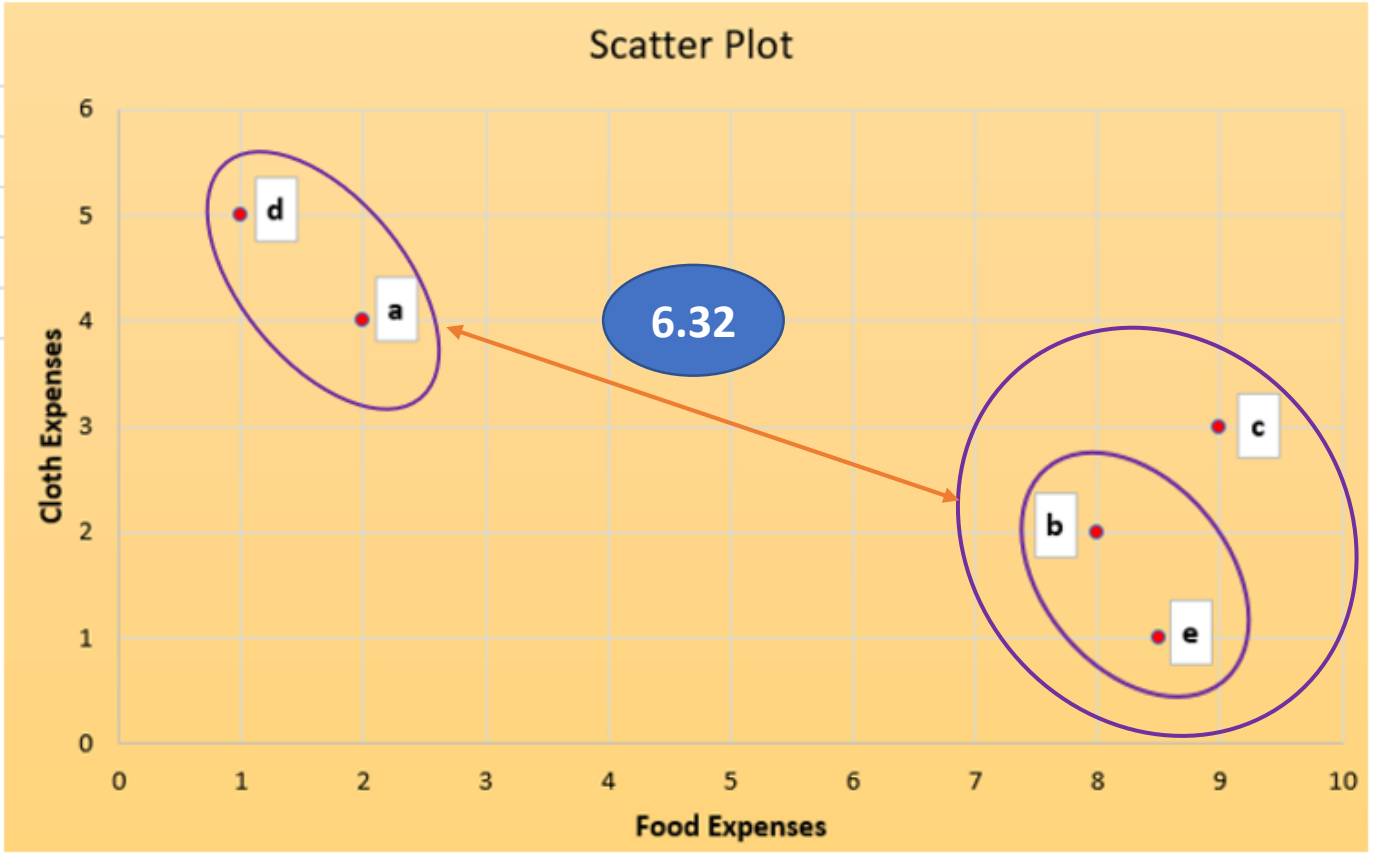
Stage 4

	a	b	c	d	e
a	0.00	6.32	7.07	1.41	7.16
b		0.00	1.41	7.62	1.12
c			0.00	8.25	2.06
d				0.00	8.50
e					0.00

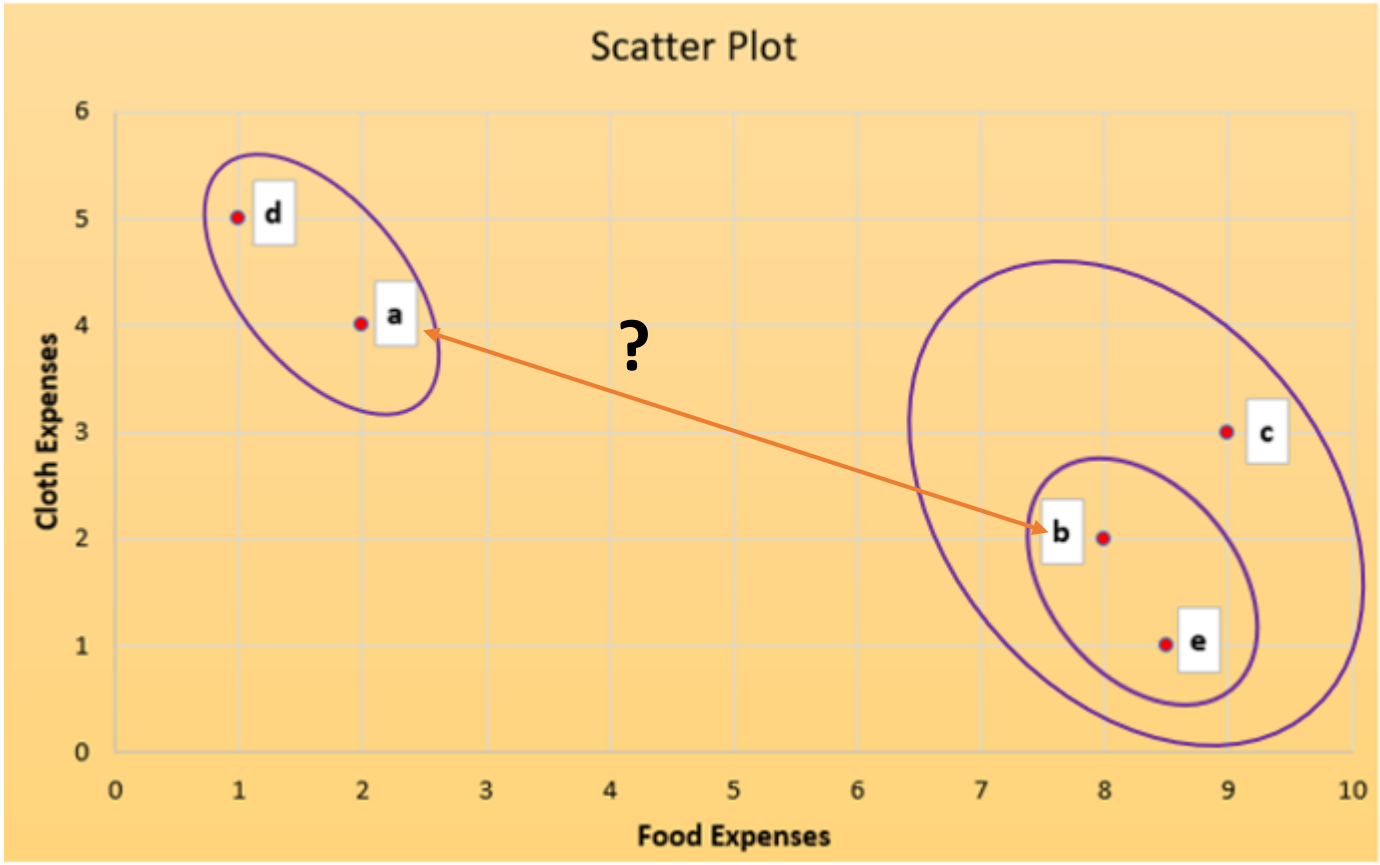
	b e c	a d
b e c	0	6.32
a d		0

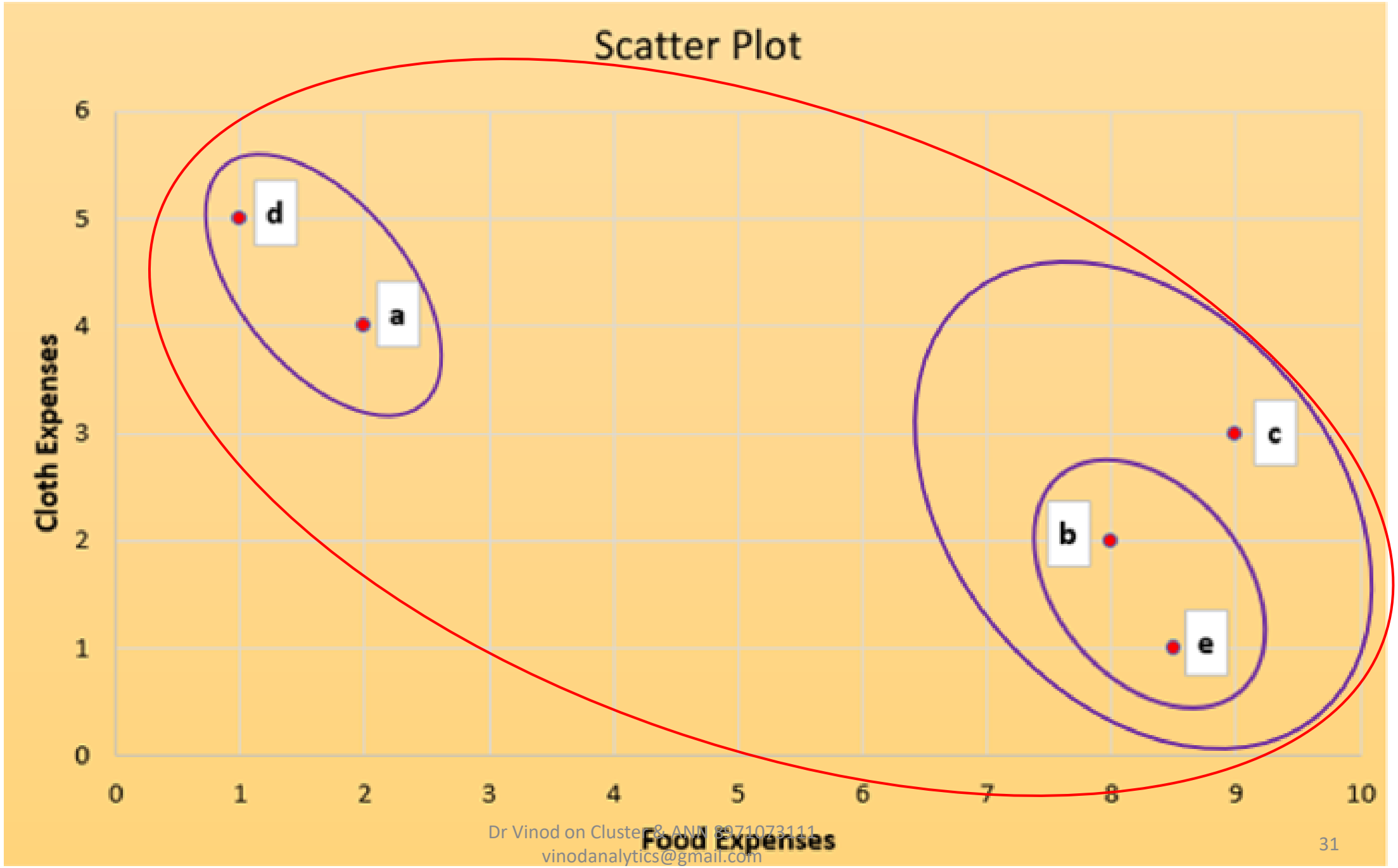
$$D(bec, ad) = min\{D(b,a), D(b,d), D(e,a), D(e,d)\ \ D(c,a), D(c,d)\}$$

$$D(bec, ad) = min\{\mathbf{6.32}, 7.62, 7.16, 8.50, 7.07, 8.25\}$$



Final round: Distance between a d and b e c




Scatter Plot
Cloth Expenses
Food Expenses
0
1
2
3
4
5
6
7
8
9
10
d
a
c
b
e

Single Linkage: Nearest Neibhbor

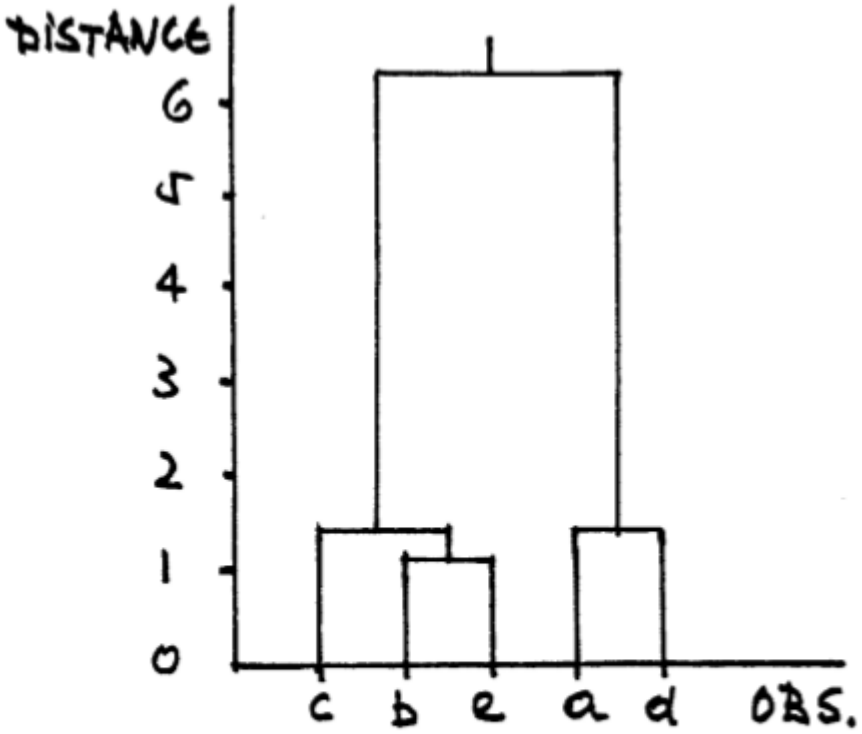


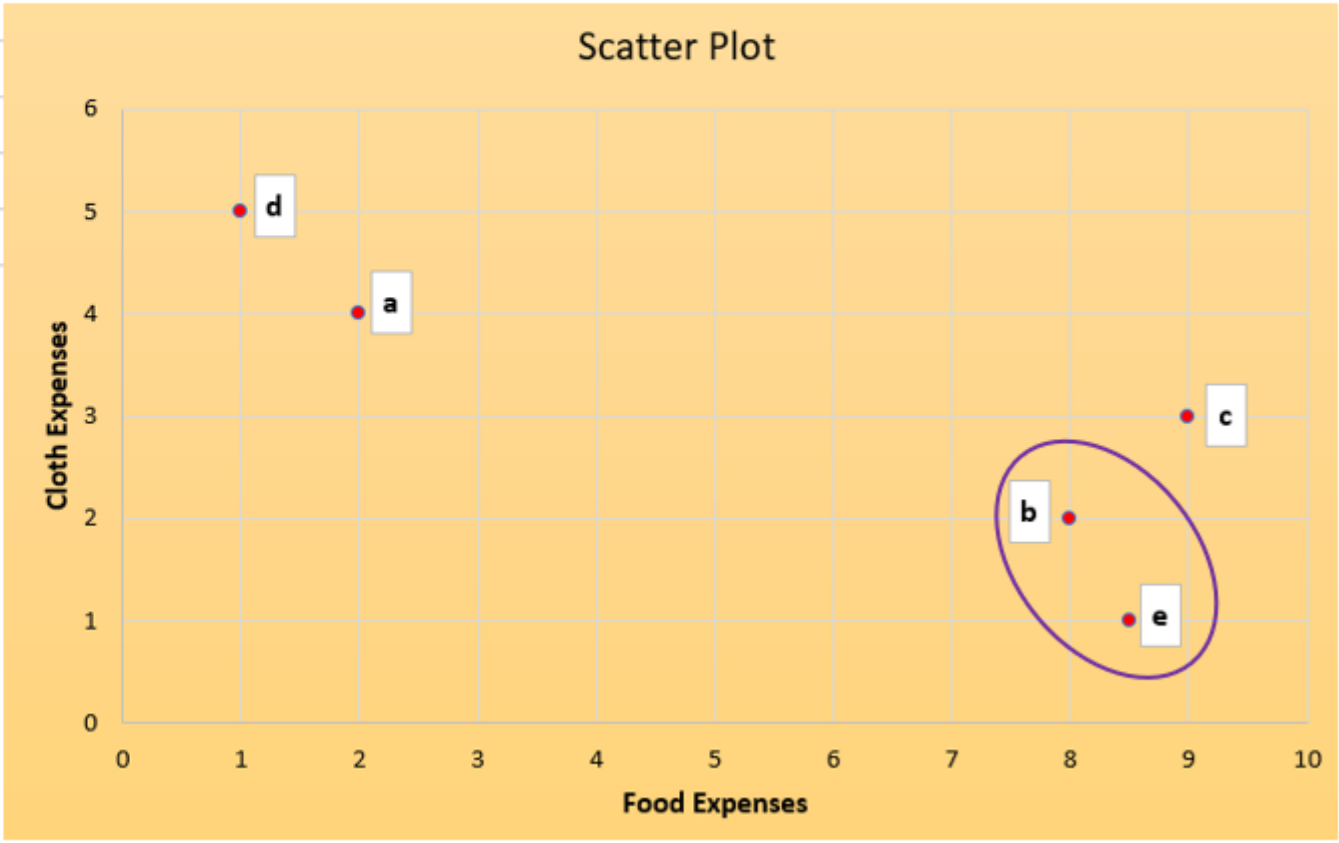
Figure 15.8
Nearest neighbor method, dendrogram

Complete Linkage: Farthest Neighbor

Stage 1: b and e can be clubbed

	a	b	c	d	e
a	0.00	6.32	7.07	1.41	7.16
b		0.00	1.41	7.62	1.12
c			0.00	8.25	2.06
d				0.00	8.50
e					0.00

Now *b* and *e* are in one cluster. How their values will be treated for finding DIST vis-à-vis another point/s?



Stage 2: Complete Linkage
Points a and d can be clubbed (same as Single Linkage)

$$D(be,a) = max\{D(b,a), D(e,a)\}$$

$$D(be,a) = max\{6.32, 7.16\}$$

$$D(be,a) = \mathbf{7.16}$$

$$D(be,c) = max\{D(b,c), D(e,c)\}$$

$$D(be,c) = max\{1.41, 2.06\}$$

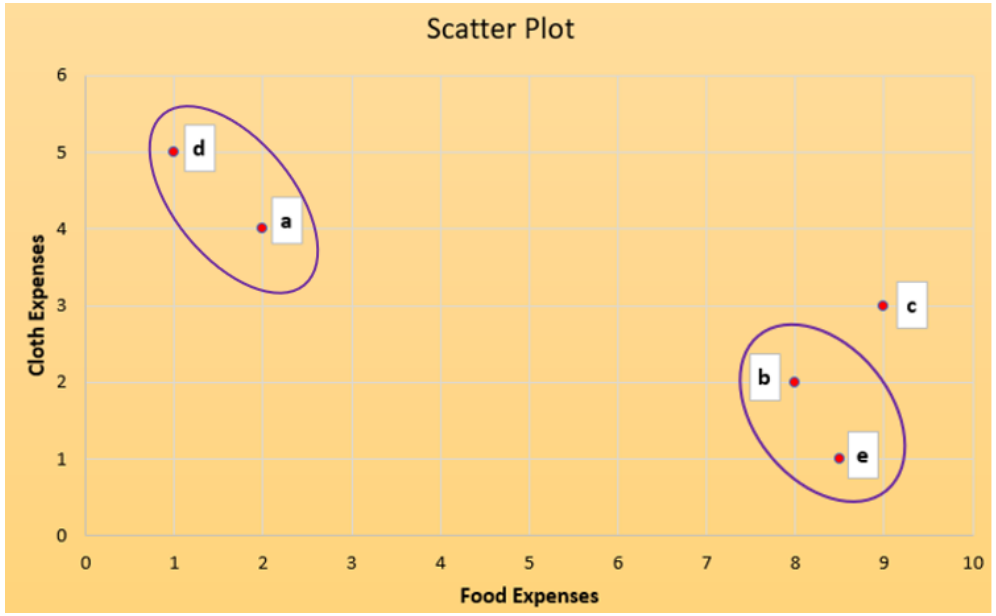
$$D(be,c) = \mathbf{2.06}$$

$$D(be,d) = max\{D(b,d), D(e,d)\}$$

$$D(be,d) = max\{7.62, 8.50\}$$

$$D(be,d) = \mathbf{8.50}$$

	b e	a	c	d
b e	**0**	***7.16***	***2.06***	***8.5***
a		**0**	**7.07**	**1.41**
c			**0**	**8.25**
d				**0**



Stage 3: Complete Linkage
Point c can be clubbed with b e (same as Single Linkage)

	b e	a	c	d
b e	0	*7.16*	*2.06*	*8.5*
a		0	7.07	1.41
c			0	8.25
d				0

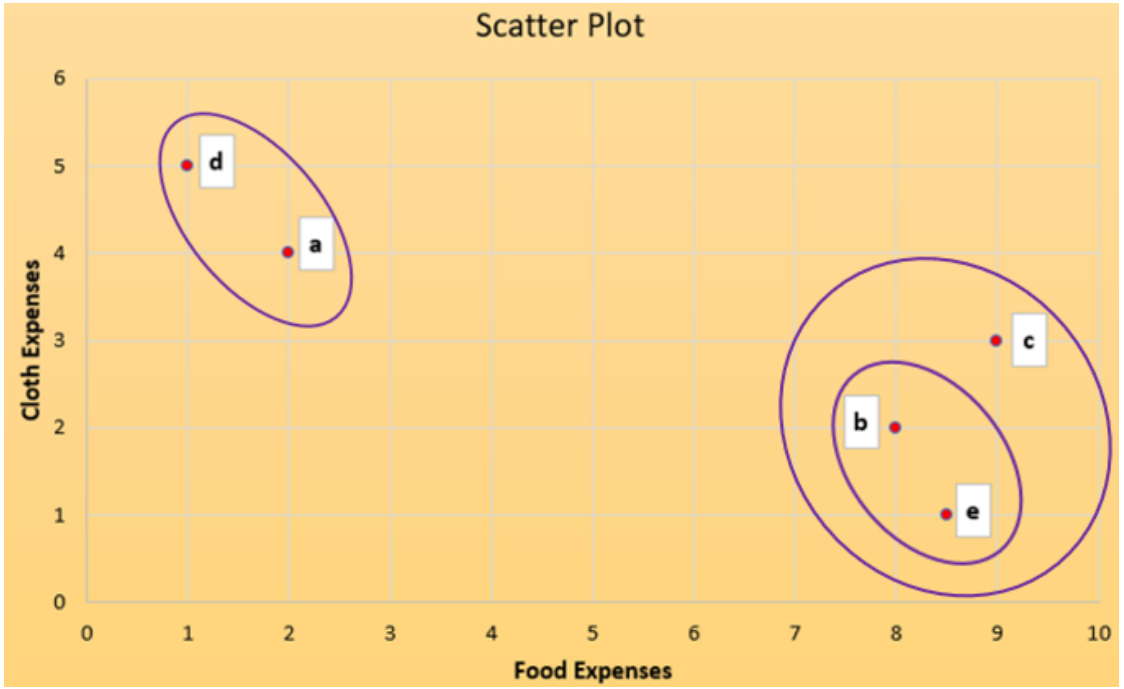
Only **D**(**be, ad**) need to be calculated
Remaining are available in above table

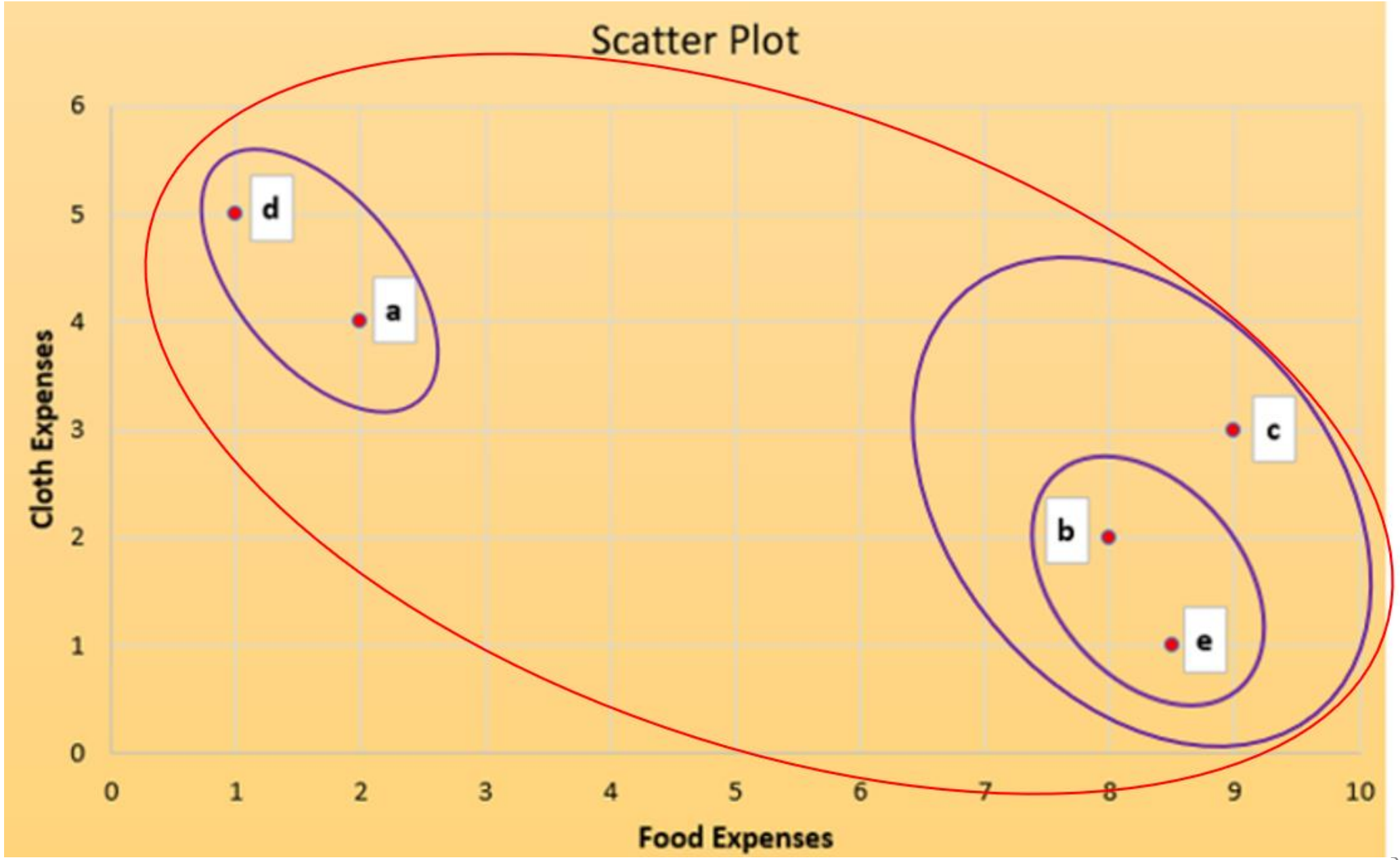
	b e	a d	c
b e	0	8.5	2.06
a d		0	8.25
c			0

$$D(be, ad) = max\{D(be, a), D(be, d)\}$$

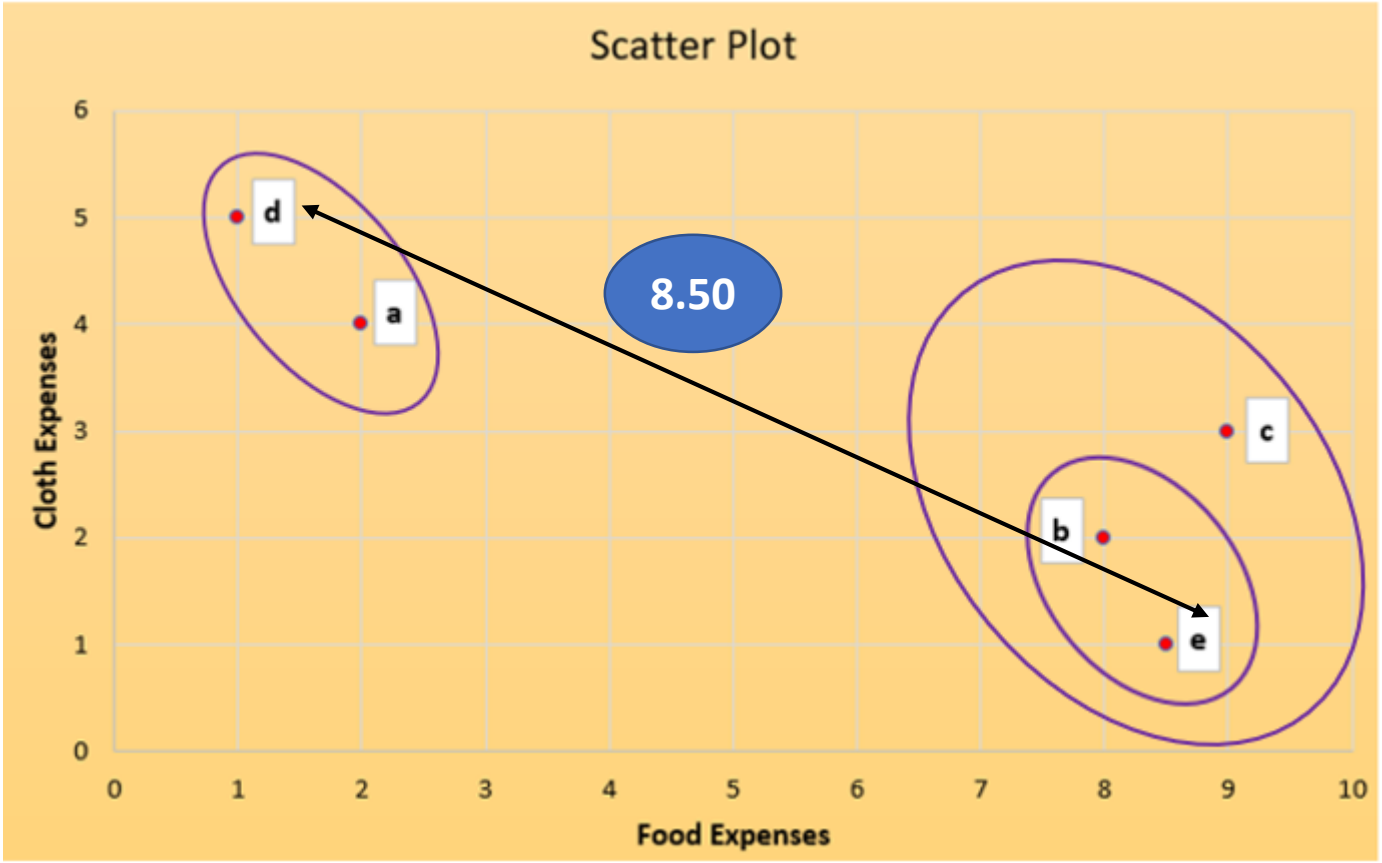
$$D(be, ad) = max\{7.16, 8.50\}$$

$$D(be, ad) = \mathbf{8.50}$$




Scatter Plot
Cloth Expenses
Food Expenses
0
1
2
3
4
5
6
7
8
9
10
d
a
c
b
e

Final round: Distance between a d and b e c



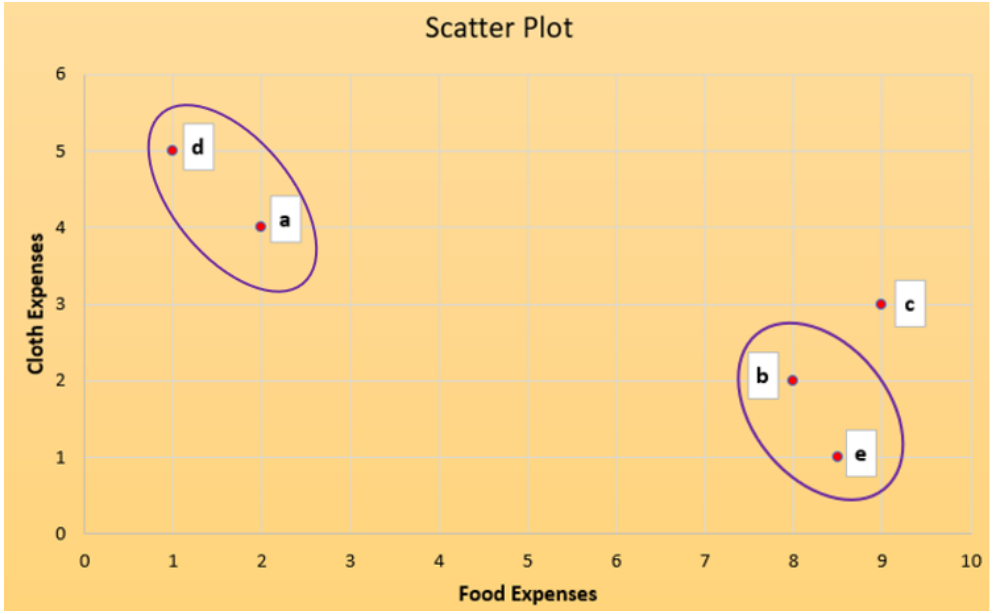
$$D(bec, ad) = max\{D(b,a), D(b,d), D(e,a), D(e,d)\ \ D(c,a), D(c,d)\}$$

$$D(bec, ad) = max\{\mathbf{6.32}, 7.62, 7.16, \mathbf{8.50}, 7.07, 8.25\}$$

Average Linkage: Compromise (Min Max)

Stage 1 & 2

	a	b	c	d	e
a	0.00	6.32	7.07	1.41	7.16
b		0.00	1.41	7.62	1.12
c			0.00	8.25	2.06
d				0.00	8.50
e					0.00



	b e	a	c	d
b e	0	6.74	1.74	8.06
a		0	7.07	1.41
c			0	8.25
d				0

First line only to be calculated (as averages), Rest from upper table

$$D(be,a) = \{D(b,a) + D(e,a)\}/2$$

$$D(be,a) = \frac{\{6.32 + 7.16\}}{2} = \mathbf{6.74}$$

$$D(be,d) = \{D(b,d) + D(e,d)\}/2$$

$$D(be,d) = \frac{\{7.62 + 8.50\}}{2} = \mathbf{8.06}$$

$$D(be,c) = \{D(b,c) + D(e,c)\}/2$$

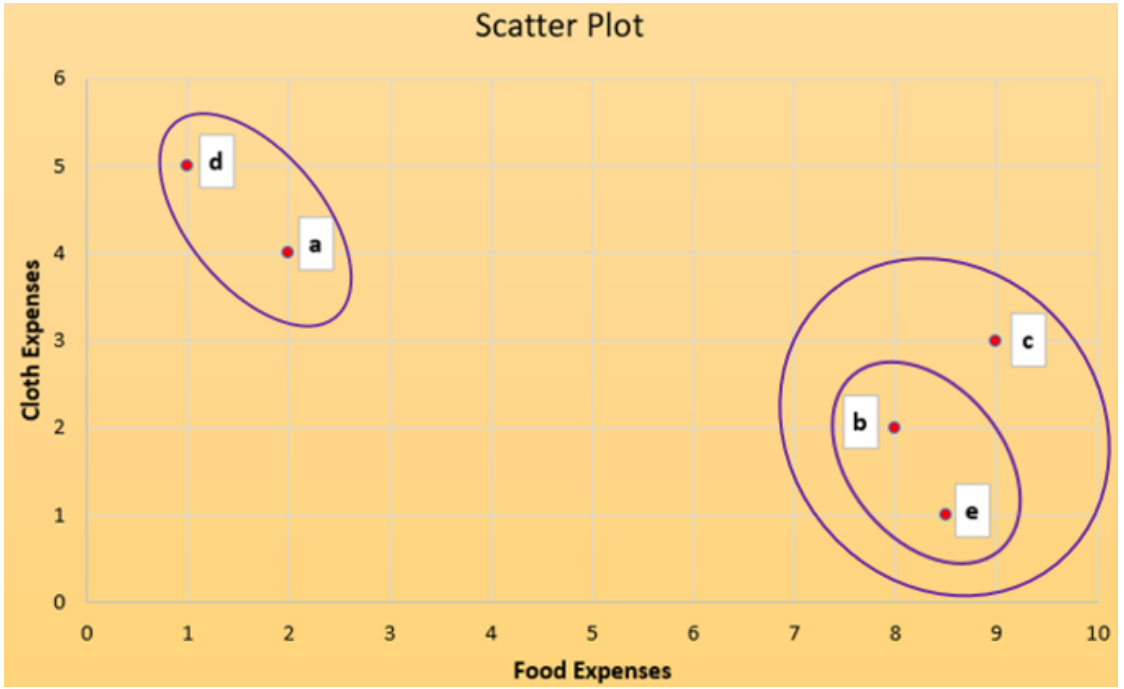
$$D(be,c) = \frac{\{1.41 + 2.06\}}{2} = \mathbf{1.74}$$

Stage 3

	b e	a	c	d
b e	**0**	**6.74**	**1.74**	**8.06**
a		**0**	**7.07**	**1.41**
c			**0**	**8.25**
d				**0**

	b e	a d	c
b e	**0**	**7.4**	**1.74**
a d		**0**	**7.66**
c			**0**

1.74 is from upper table, rest two were calculated



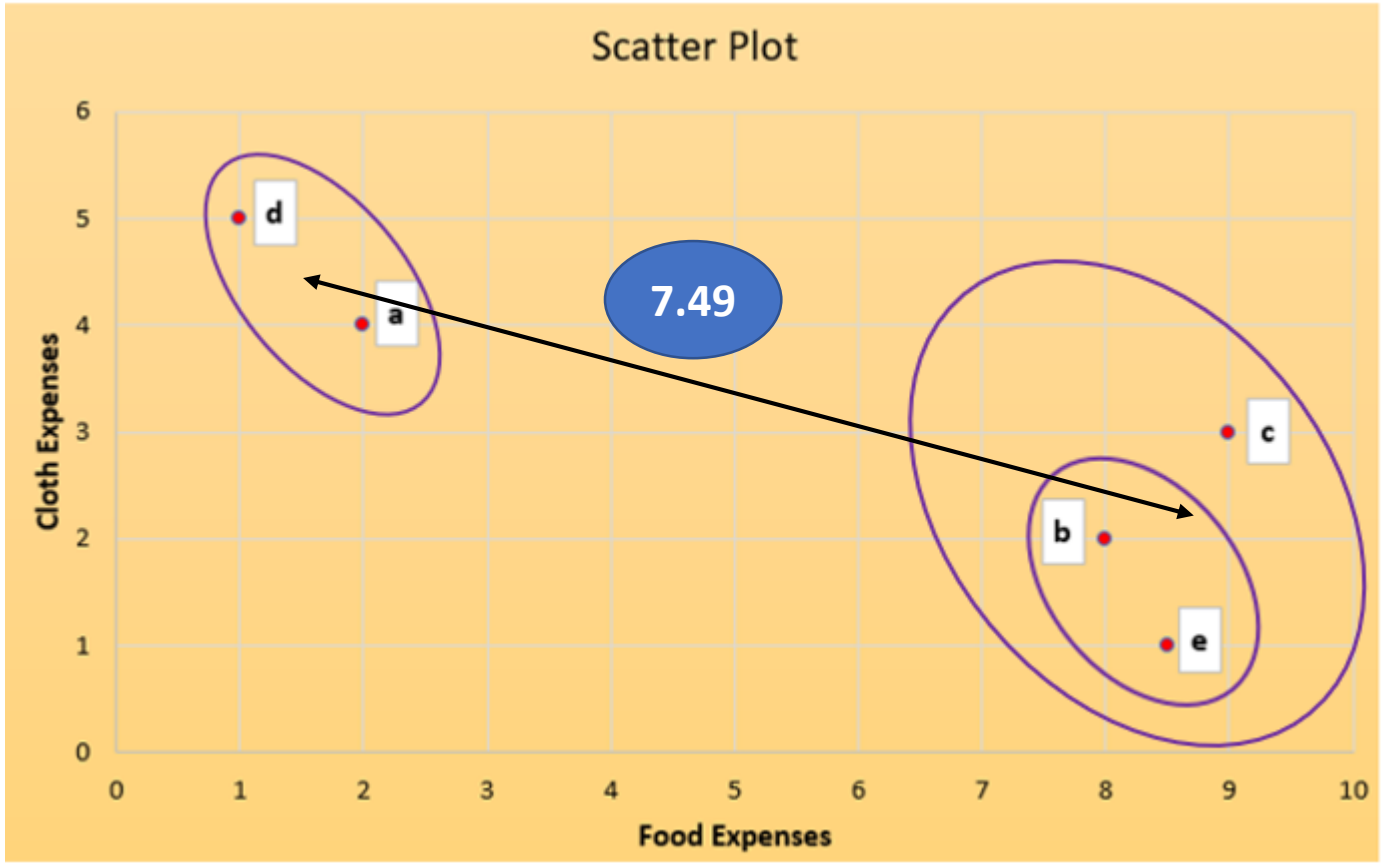
$$D(be, ad) = \{D(be, a) + D(be, d)\}/2$$

$$D(be, ad) = \frac{\{6.74 + 8.06\}}{2} = \mathbf{7.4}$$

$$D(ad, c) = \{D(a, c) + D(d, a)\}/2$$

$$D(ad, c) = \frac{\{7.07 + 8.25\}}{2} = \mathbf{7.66}$$

Final round: Distance between a d and b e c



$$D(bec, ad) = average\{D(b,a), D(b,d), D(e,a), D(e,d)\ \ D(c,a), D(c,d)\}$$

$$D(bec, ad) = average\{6.32, 7.62, 7.16, 8.50, 7.07, 8.25\} = 7.49$$

Ward Linkage: Finds centroids and then SSE

Distance Matrix

Ward's Method

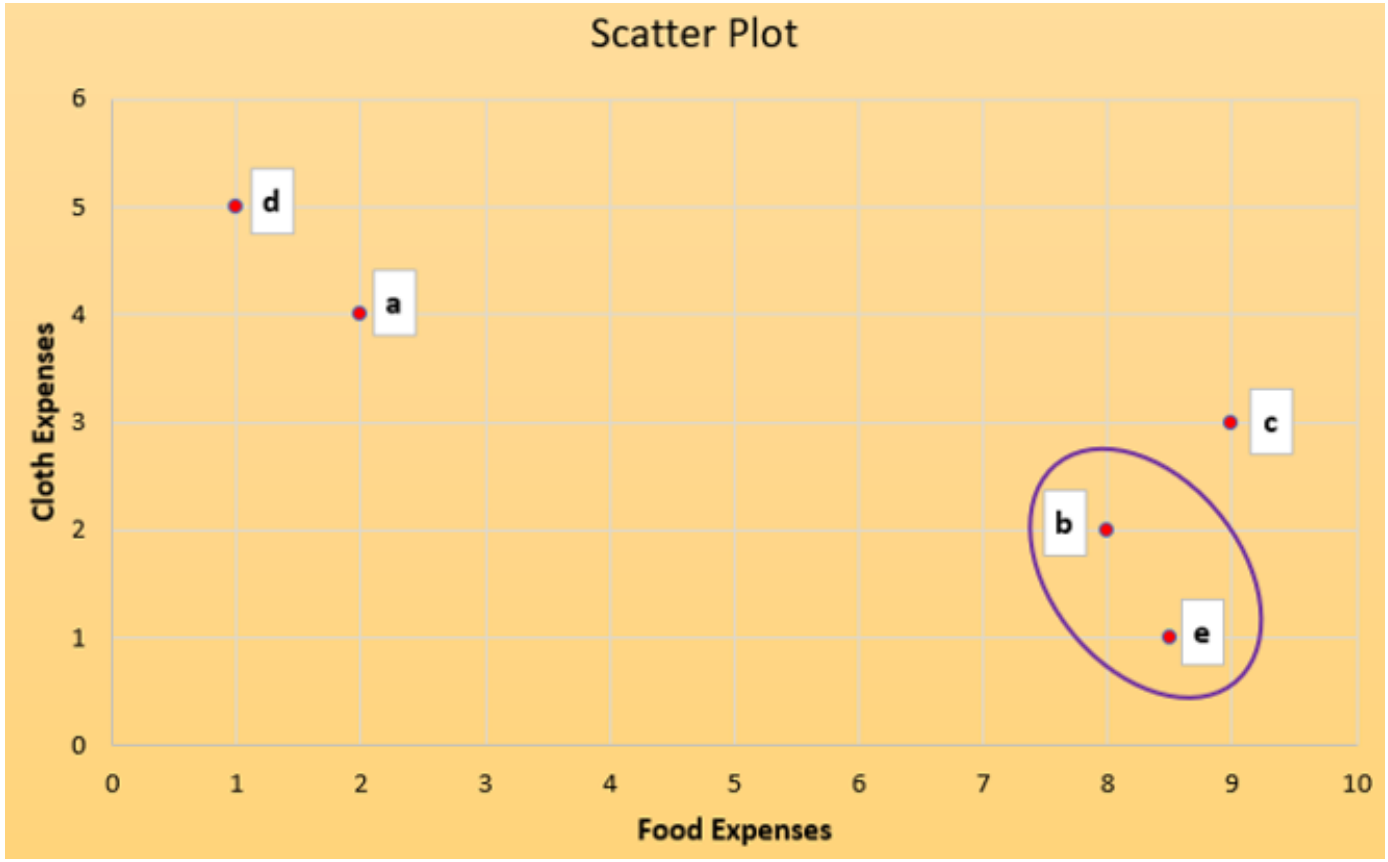
Ward's method means calculating the incremental sum of squares. *Half square Euclidean distance* is the only distance measure that can be used with this clustering method. Therefore, the distance measure is automatically set to *Half square Euclidean distance* when Ward's method is selected.

	a	b	c	d	e	stage 1
a	**0.00**	**20.00**	**25.00**	**1.00**	**25.63**	
b		**0.00**	**1.00**	**29.00**	**0.63**	
c			**0.00**	**34.00**	**2.13**	
d				**0.00**	**30.13**	
e					**0.00**	

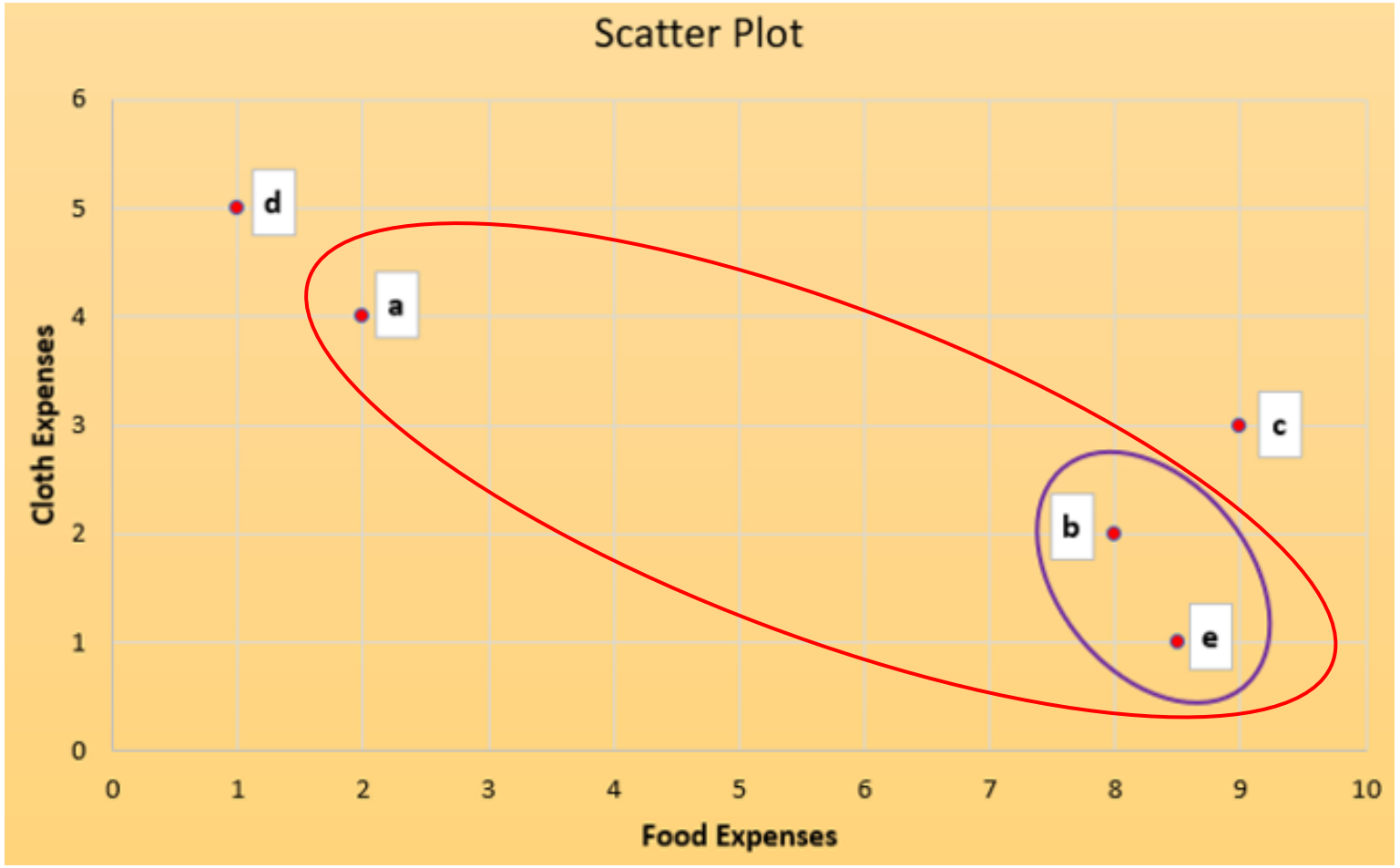
Euclidean Distances

	b e	a	c	d
b e	**0**	**6.74**	**1.74**	**8.06**
a		**0**	**7.07**	**1.41**
c			**0**	**8.25**
d				**0**

Stage 1


Scatter Plot
Cloth Expenses
Food Expenses
0
1
2
3
4
5
6
7
8
9
10
d
a
c
b
e

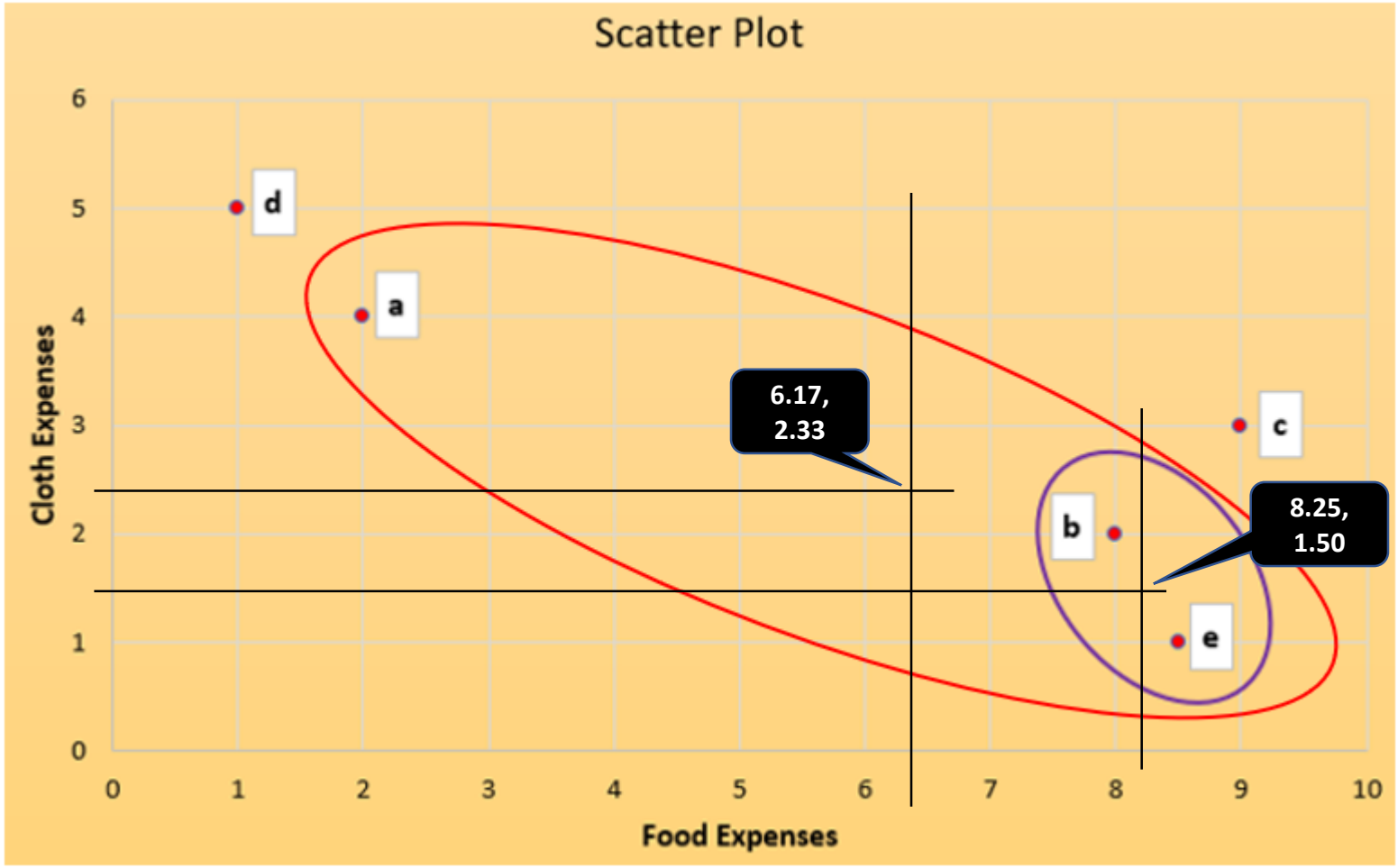
Distance Between (be) and a



Distance Between (be) and a
First find centroids

Centroid for cluster (bea)		
	X	Y
b	8	2
e	8.5	1
a	2	4
	6.17	**2.33**

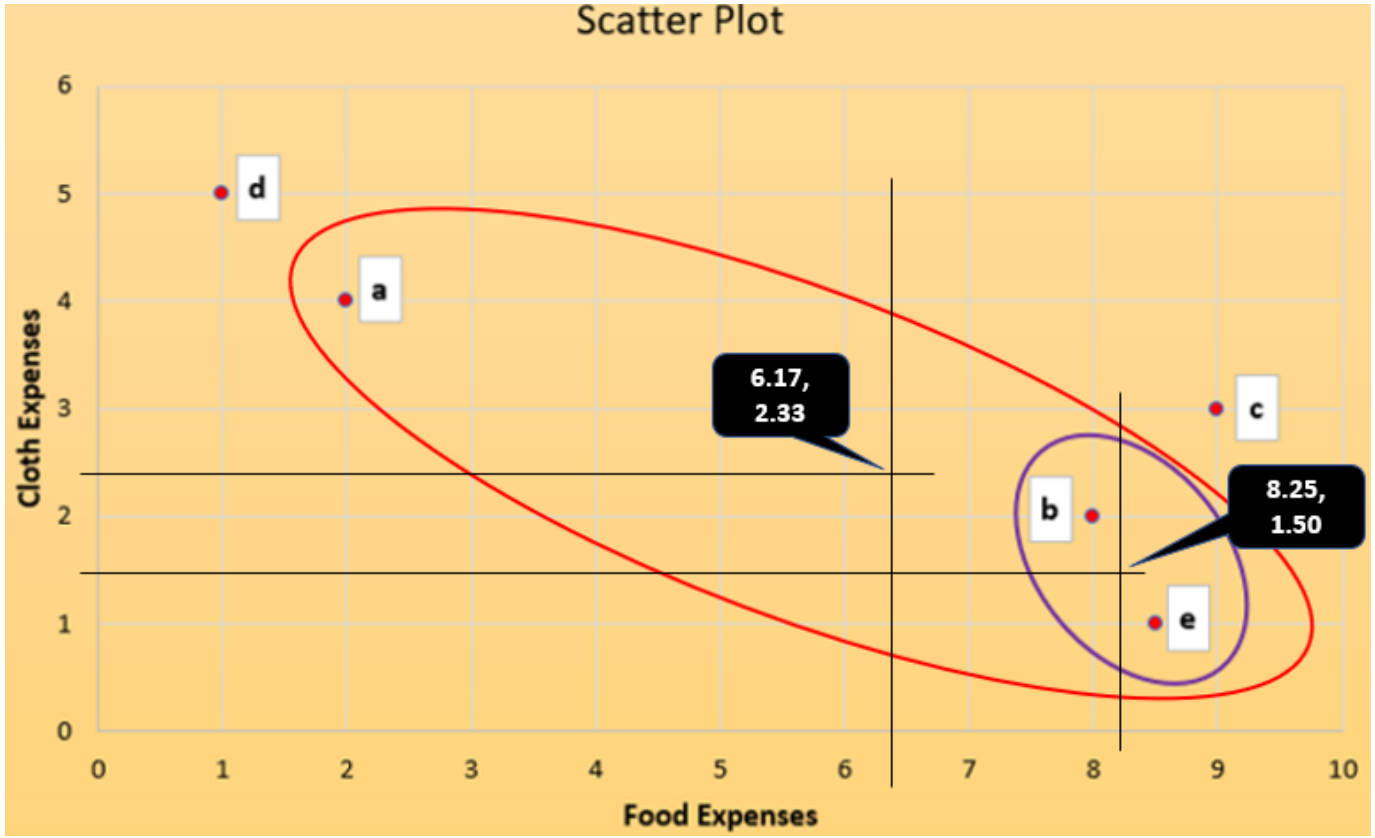
Centroid for cluster (be)		
	X	Y
b	8	2
e	8.5	1
	8.25	**1.50**



Distance Between (be) and a

Centroid for cluster (bea)		
	X	Y
b	8	2
e	8.5	1
a	2	4
	6.17	**2.33**
SSE(bea)	**30.83**	

Centroid for cluster (be)		
	X	Y
b	8	2
e	8.5	1
	8.25	**1.50**
SSE(be)	**0.63**	



$$dist_{(be,a)} = SSE_{(bea)} - SSE_{be} - SSE_a$$

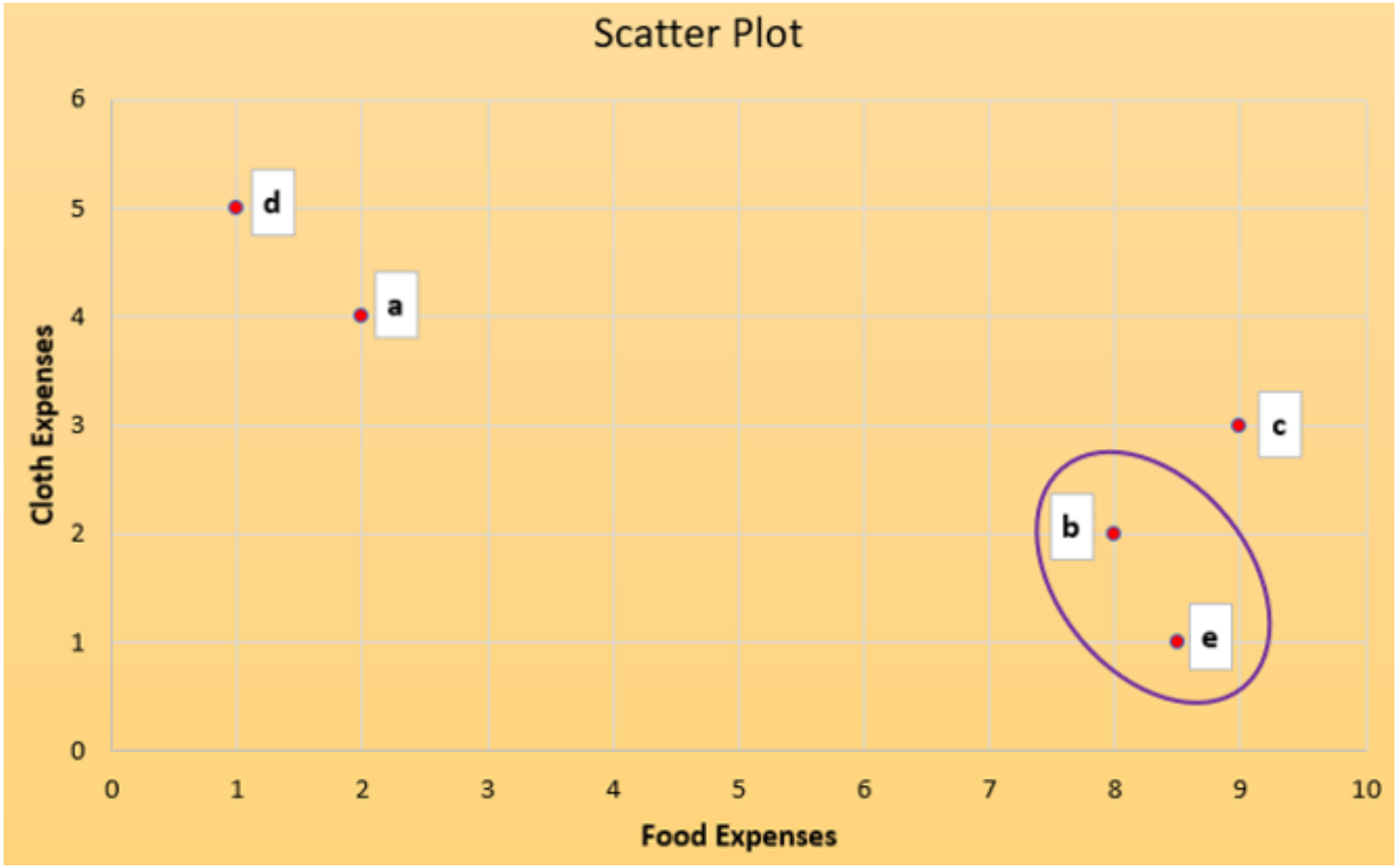
$$dist_{(be,a)} = 30.83 - 0.63 - 0 = 30.20$$

Distance Between (be) and d
First find centroids then find SSEs

Centroid for cluster (bed)		
	X	Y
b	8	2
e	8.5	1
d	1	5
	5.83	**2.67**
SSE(bed)	**43.83**	

Centroid for cluster (be)		
	X	Y
b	8	2
e	8.5	1
	8.25	**1.50**
SSE(be)	**0.63**	

dist(be,d)	**43.21**

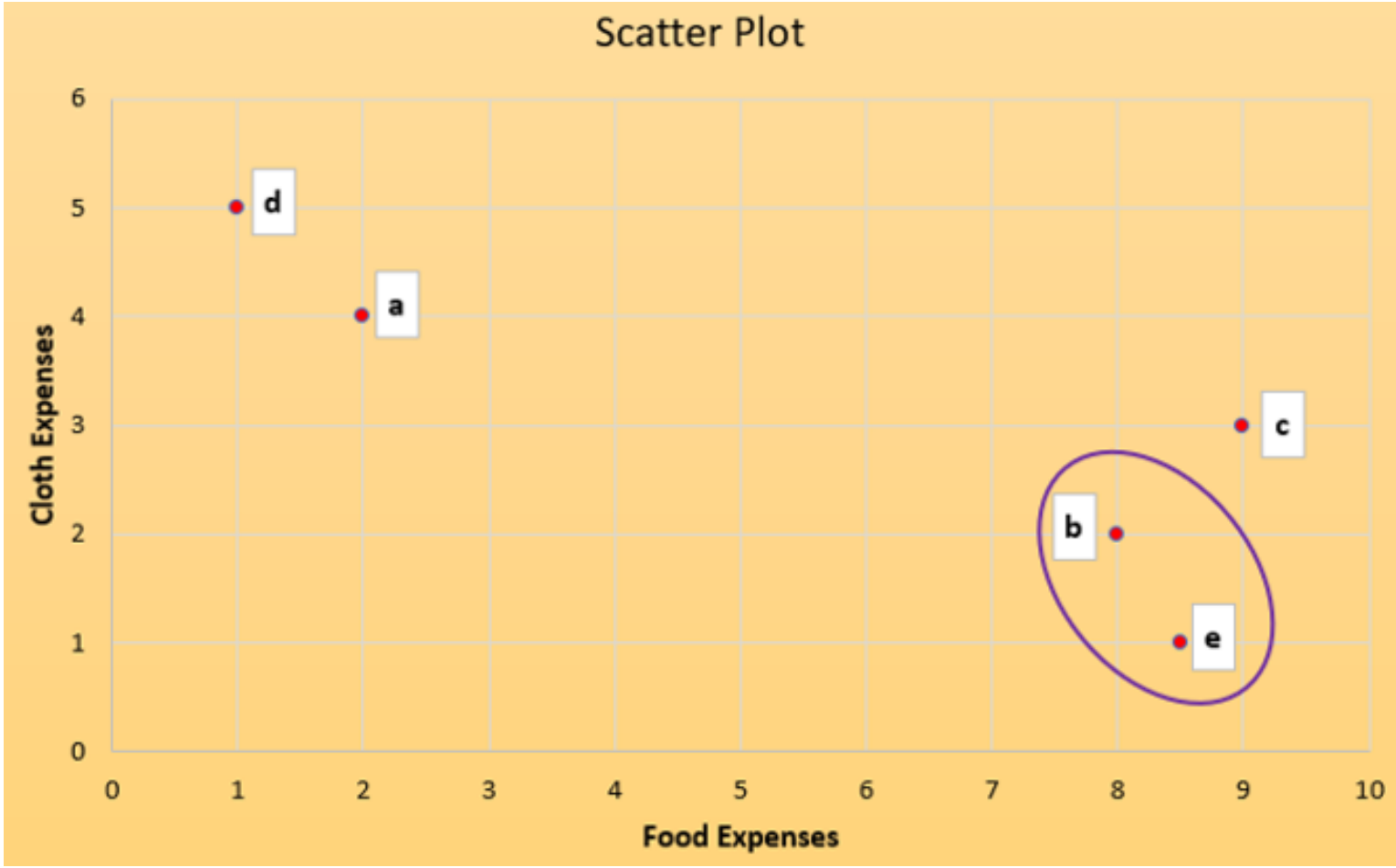


Distance Between (be) and c
First find centroids then find SSEs

Centroid for cluster (bec)		
	X	Y
b	8	2
e	8.5	1
c	9	3
	8.50	**2.00**
SSE(bec)	**2.50**	

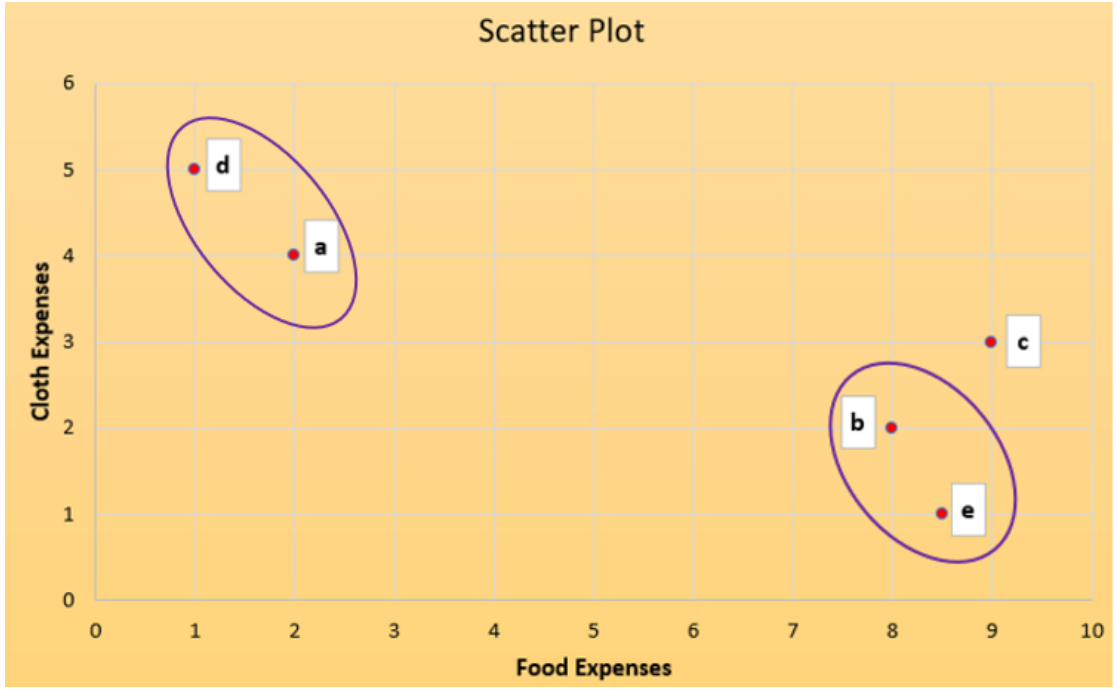
Centroid for cluster (be)		
	X	Y
b	8	2
e	8.5	1
	8.25	**1.50**
SSE(be)	**0.63**	

dist(be,c)	**1.88**



Stage 2

	b e	a	c	d	stage 2
b e	**0**	**30.21**	**1.88**	**43.21**	
a		**0**	**25.00**	**1.00**	
c			**0**	**34.00**	
d				**0**	



Lets make it

```
In [1]: # Jesus is my Saviour!

In [2]: import os

In [3]: os.chdir('C:\\Users\\Dr Vinod\\Desktop\\WD_python')

In [4]: # our exported file will appear here

In [5]: import pandas as pd

In [6]: import numpy as np

In [7]: import matplotlib.pyplot as plt

In [8]: from sklearn.preprocessing import normalize
```

Data: Wholesale customers data

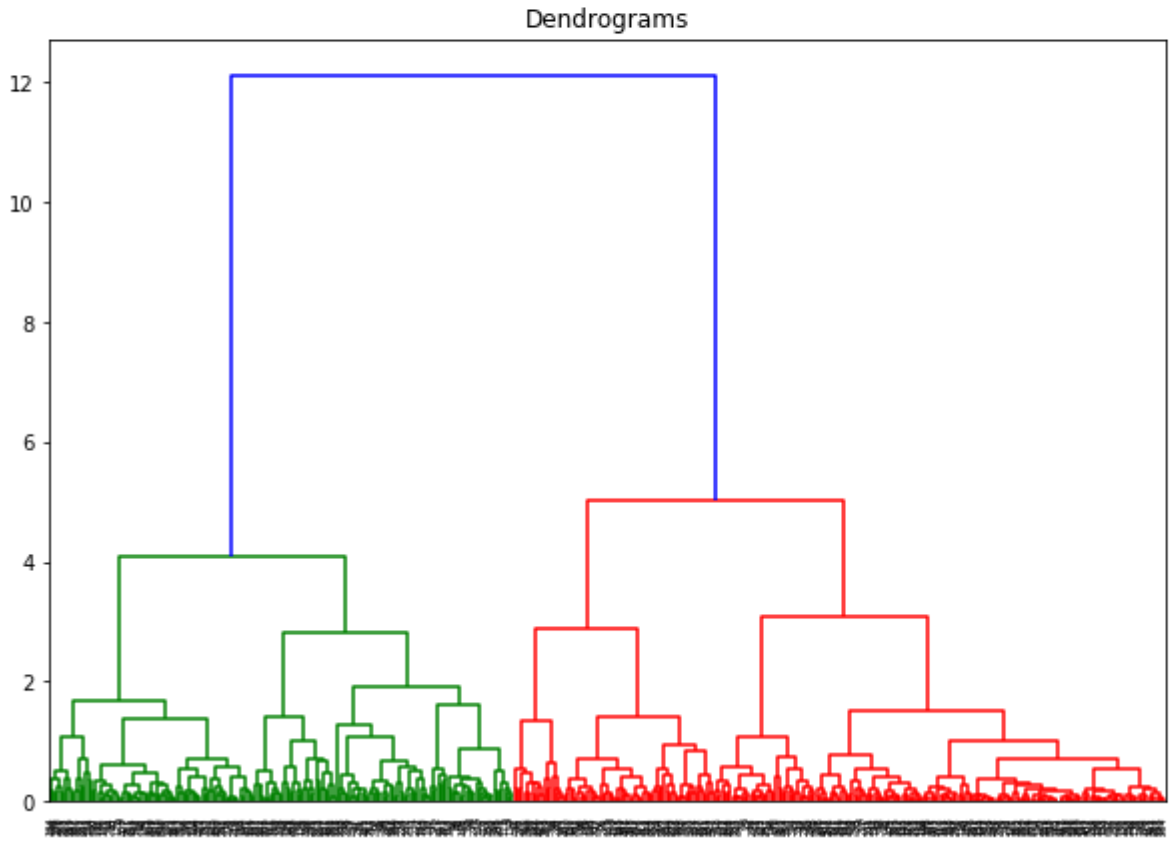
```
In [9]: data = pd.read_csv("C:/Users/Dr Vinod/Desktop/DataSets1/Wholesale customers
data.csv")

In [10]: data = pd.DataFrame(data)

In [11]: data.head(3)
Out[11]:
   Channel  Region  Fresh  Milk  Grocery  Frozen  Detergents_Paper  Delicassen
0        2       3  12669  9656     7561     214              2674        1338
1        2       3   7057  9810     9568    1762              3293        1776
2        2       3   6353  8808     7684    2405              3516        7844
```

Scale the data

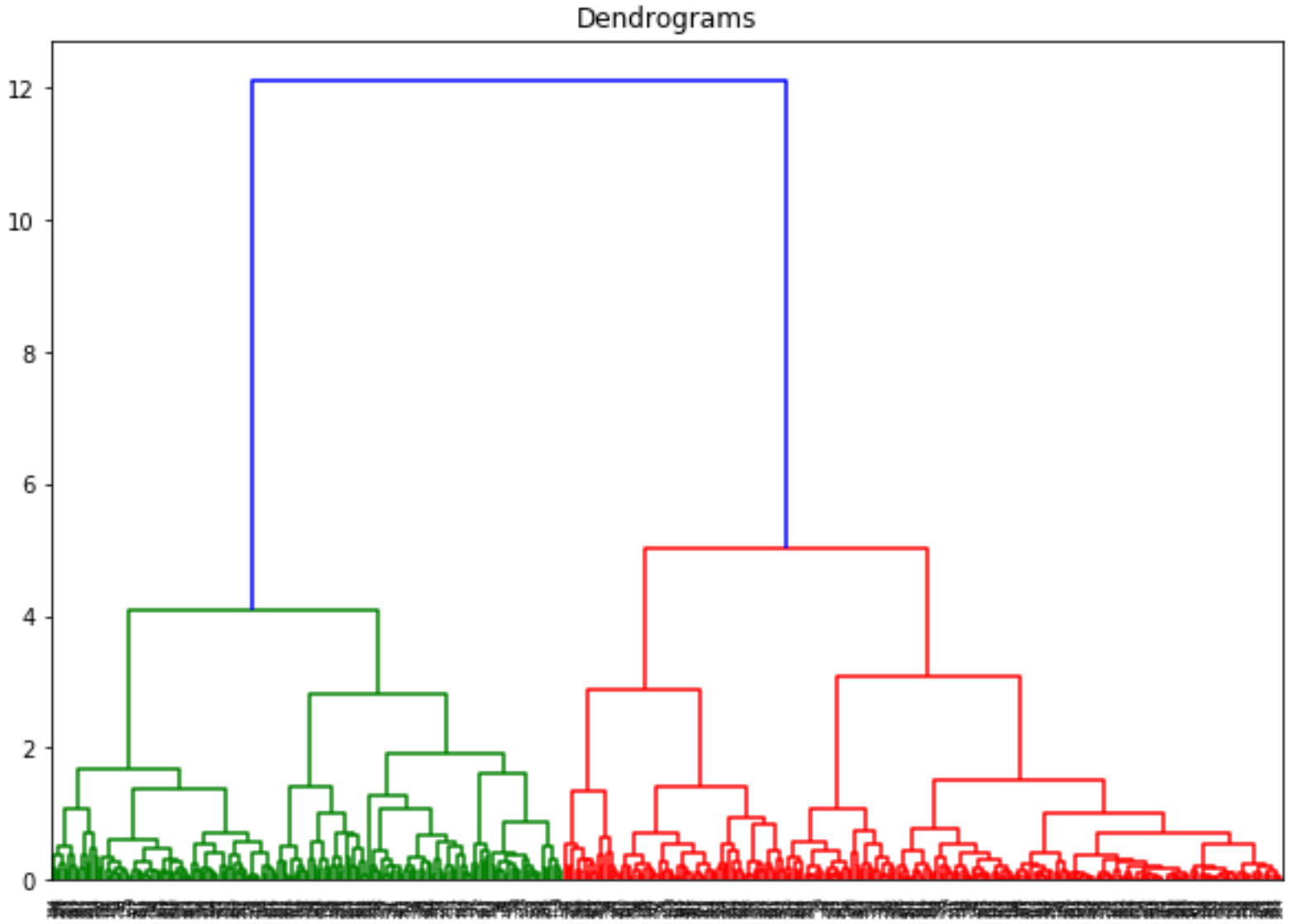
```
In [12]: data_scaled = normalize(data)

In [13]: data_scaled = pd.DataFrame(data_scaled, columns = data.columns)

In [14]: data.head(data_scaled)
Traceback (most recent call last):
```

Dendrogram

```
In [15]: import scipy.cluster.hierarchy as shc

In [16]: plt.figure(figsize = (10,7))
    ...: plt.title("Dendrograms")
    ...: dend = shc.dendrogram(shc.linkage(data_scaled, method = 'ward'))
```

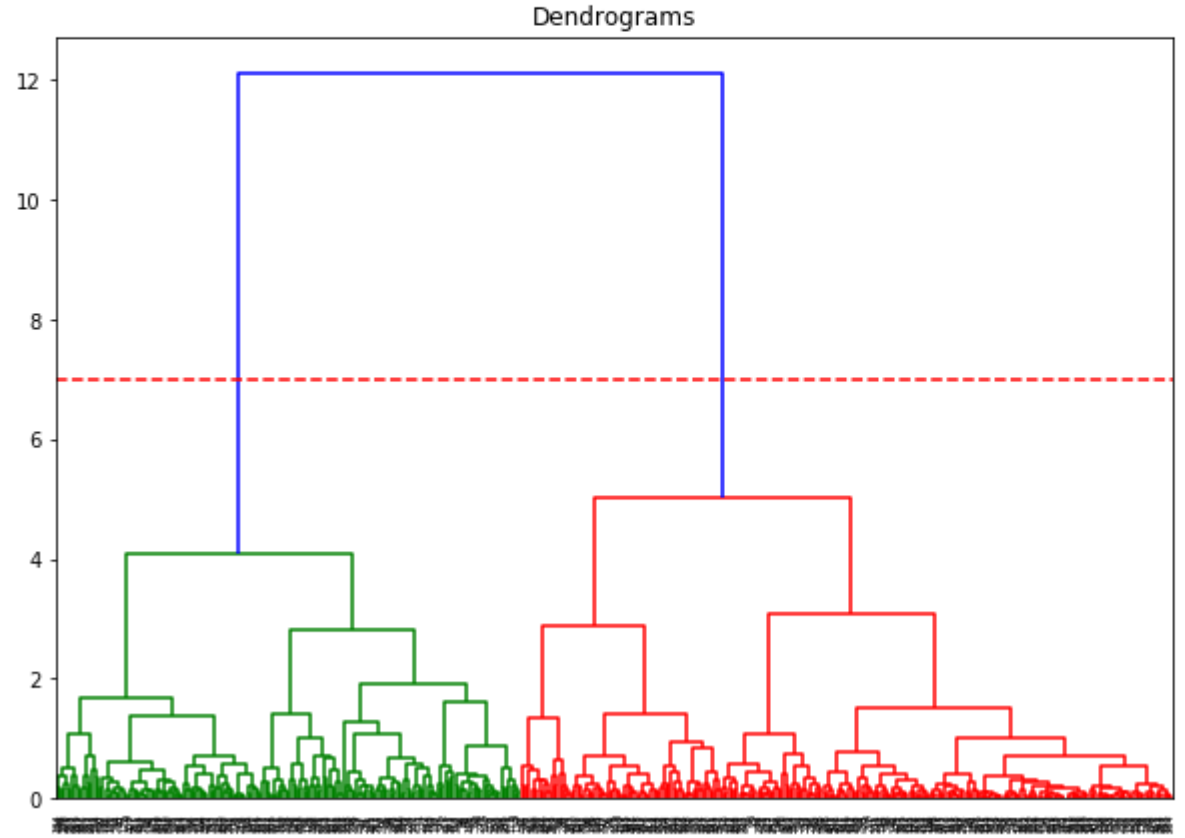


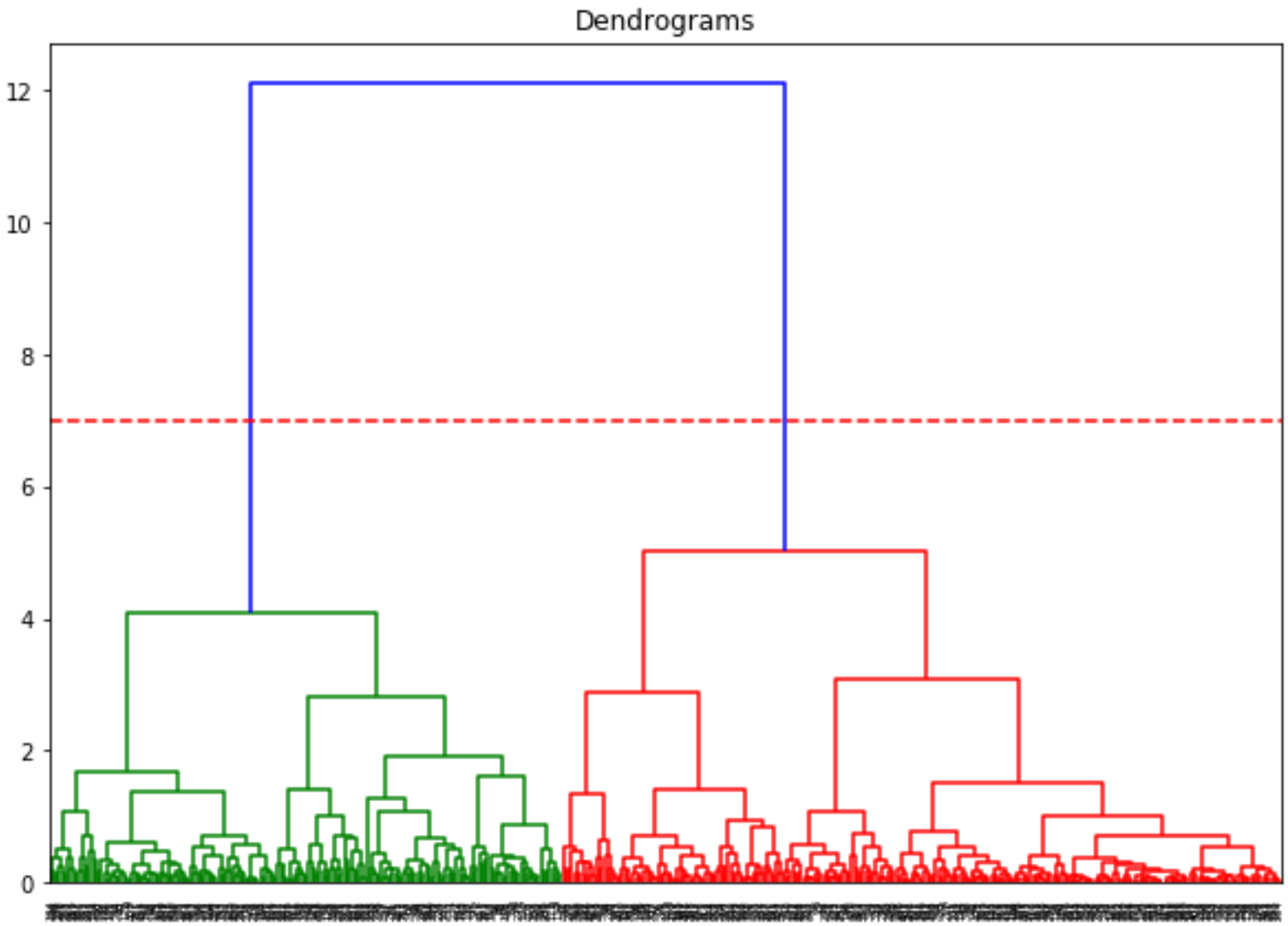
Dr Vinod on Cluster & ANN 8971073111
vinodanalytics@gmail.com


Dendrograms
12
10
8
6
4
2
0

2 clusters

```
In [17]: plt.figure(figsize = (10,7))
    ...: plt.title("Dendrograms")
    ...: dend = shc.dendrogram(shc.linkage(data_scaled, method = 'ward'))
    ...: plt.axhline(y=7, color = "r", linestyle = '--')
Out[17]: <matplotlib.lines.Line2D at 0x2242398ec50>
```



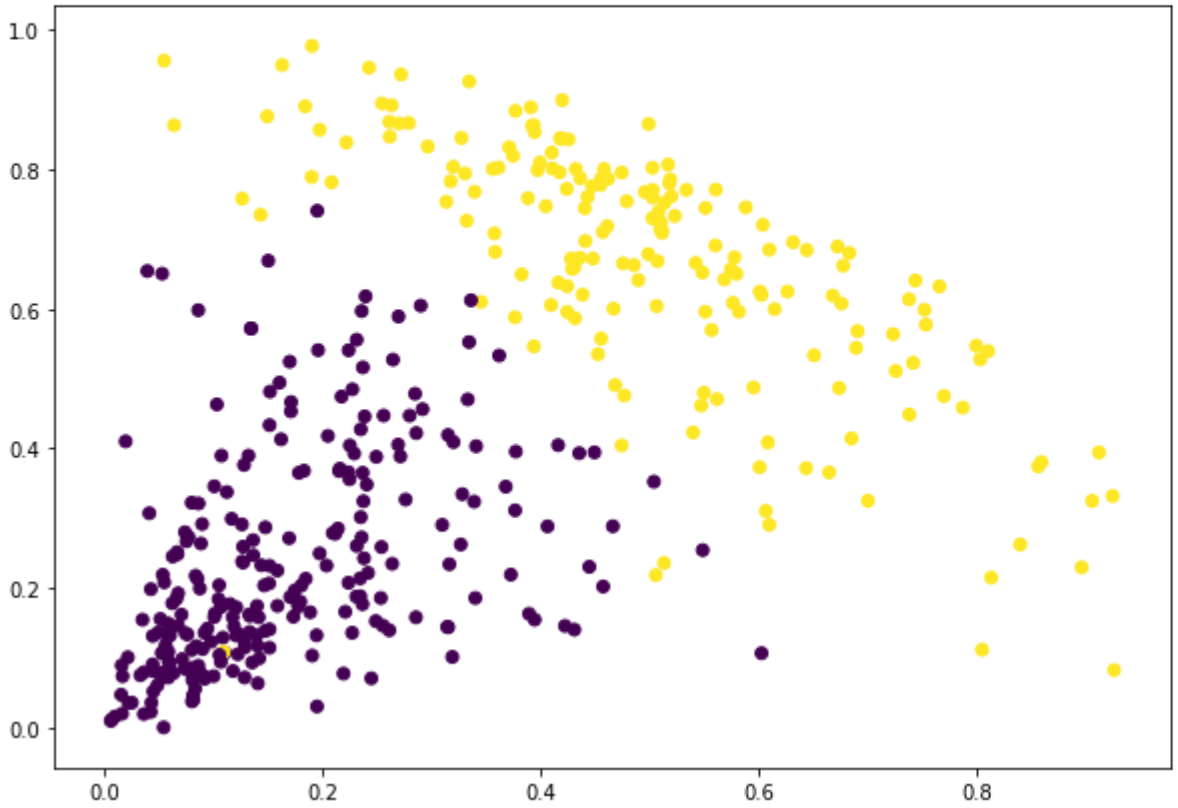

Dendrograms
12
10
8
6
4
2
0

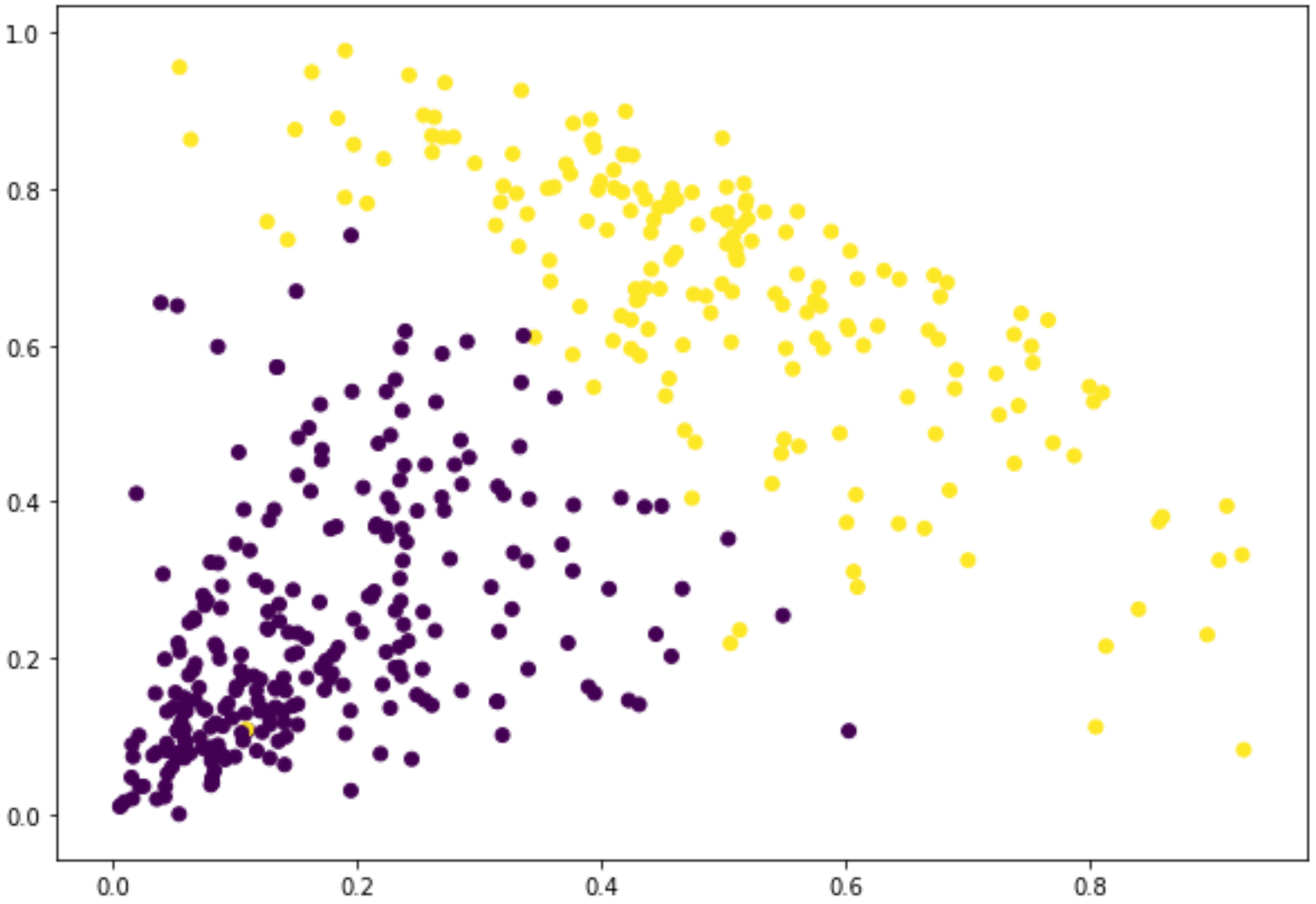
Lets make clusters

```
In [18]: #_______lets apply Hierarchical clustering for 2 clusters

In [19]: from sklearn.cluster import AgglomerativeClustering

In [20]: cluster = AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean',
linkage = 'ward')

In [21]: cluster.fit_predict(data_scaled)
Out[21]:
array([1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
```

Plot

```
In [22]: plt.figure(figsize = (10,7))
    ...: plt.scatter(data_scaled['Milk'], data_scaled['Grocery'], c=cluster.labels_)
    ...:
Out[22]: <matplotlib.collections.PathCollection at 0x22423e25898>
```



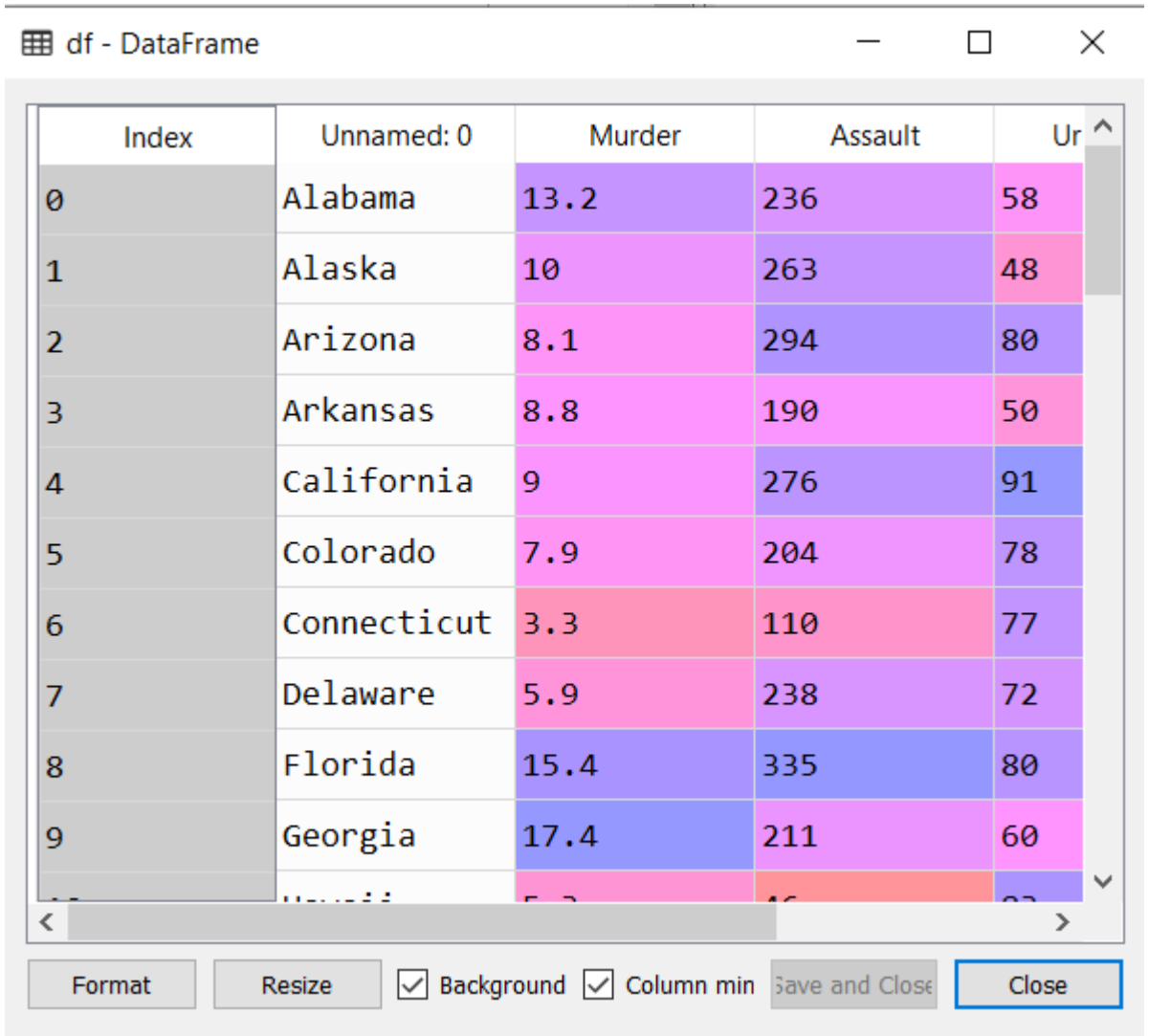

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8

K-means

Libraries

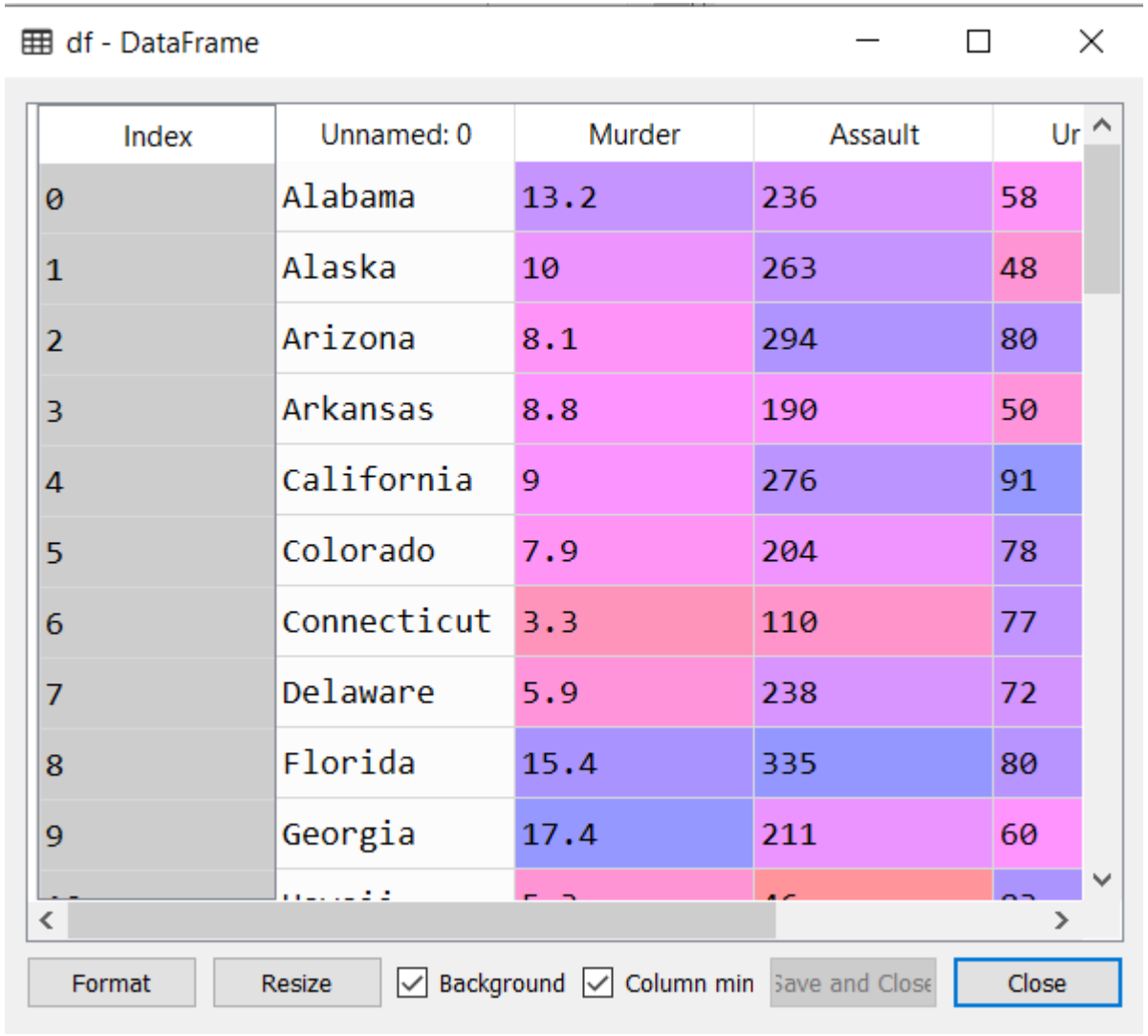
```
In [1]: # importing needed libraries

In [2]: import pandas as pd

In [3]: import matplotlib.pyplot as plt

In [4]: from sklearn import preprocessing


In [5]: from sklearn.preprocessing import scale

In [6]: from sklearn.preprocessing import StandardScaler

In [7]: from sklearn.cluster import KMeans

In [8]: import seaborn as sns
```

```
In [9]: df = pd.read_csv("C:/Users/Dr Vinod/Desktop/DataSets1/USArrests.csv")

In [10]: df = pd.DataFrame(df)
```



Index	Unnamed: 0	Murder	Assault	Ur
0	Alabama	13.2	236	58
1	Alaska	10	263	48
2	Arizona	8.1	294	80
3	Arkansas	8.8	190	50
4	California	9	276	91
5	Colorado	7.9	204	78
6	Connecticut	3.3	110	77
7	Delaware	5.9	238	72
8	Florida	15.4	335	80
9	Georgia	17.4	211	60

USArrests Data



Index	Unnamed: 0	Murder	Assault	Ur
0	Alabama	13.2	236	58
1	Alaska	10	263	48
2	Arizona	8.1	294	80
3	Arkansas	8.8	190	50
4	California	9	276	91
5	Colorado	7.9	204	78
6	Connecticut	3.3	110	77
7	Delaware	5.9	238	72
8	Florida	15.4	335	80
9	Georgia	17.4	211	60

Bring state's name as row names

```
In [11]: # Naming the 1st Unnamed column as States

In [12]: df.rename(columns={'Unnamed: 0':'States'}, inplace=True)

In [13]: # Setting States column as index

In [14]: df.set_index('States', inplace=True)

In [15]: df
Out[15]:
```

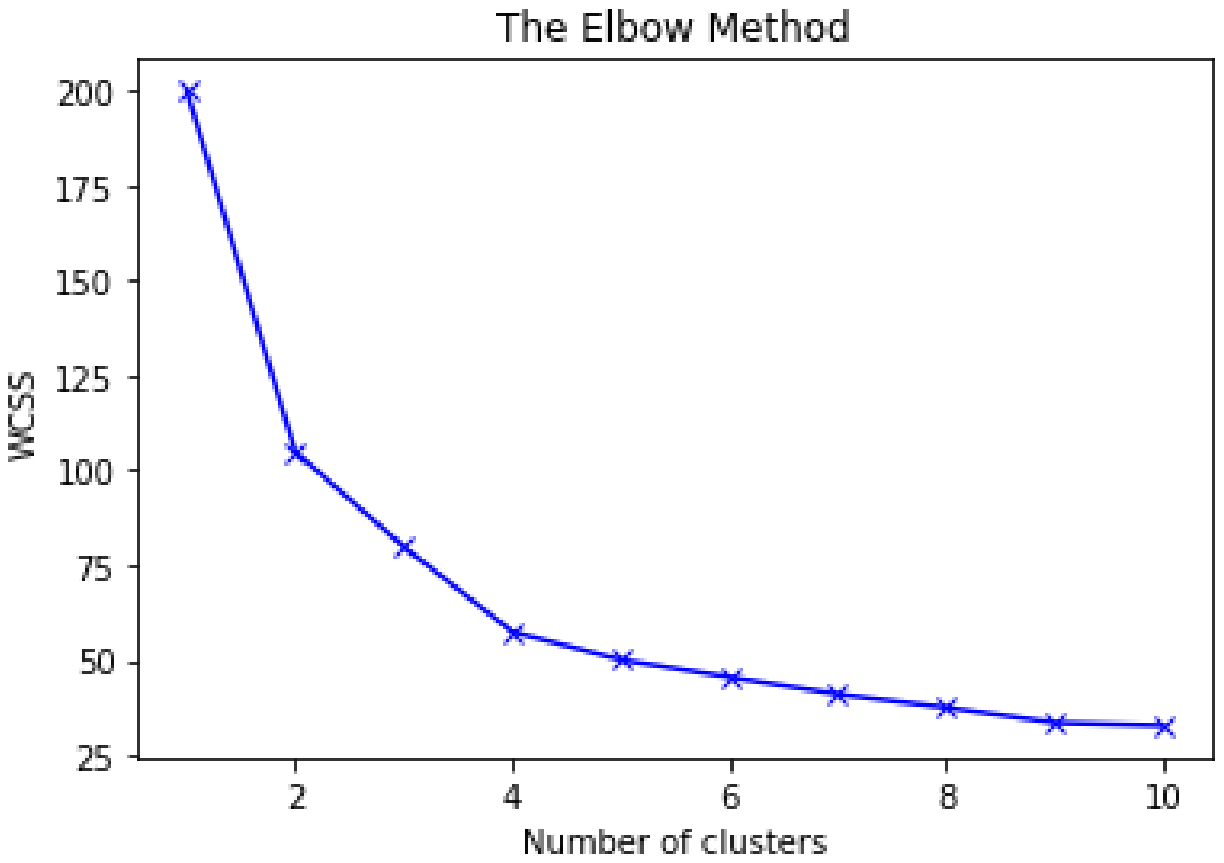
States	Murder	Assault	UrbanPop	Rape
Alabama	13.2	236	58	21.2
Alaska	10.0	263	48	44.5
Arizona	8.1	294	80	31.0
Arkansas	8.8	190	50	19.5
California	9.0	276	91	40.6

Scaling of variables

```
In [16]: # Creating X variable from df without States

In [17]: X = df[['Murder', 'Assault', 'Rape', 'UrbanPop']]

In [18]: # Scaling X variable

In [19]: scaler = StandardScaler()

In [20]: X_scaled = scaler.fit_transform( X )
```

Elbow Plot

```
In [21]: # Plotting for optimum nos of clusters

In [22]: plt.figure(figsize=(10, 8))
Out[22]: <Figure size 720x576 with 0 Axes><Figure size 720x576 with 0 Axes>

In [23]: wcss = []

In [24]: #____below in one block

In [25]: for i in range(1, 11):
    ...:     kmeans = KMeans(n_clusters = i, init = 'random', random_state = 42)
    ...:     kmeans.fit(X_scaled)
    ...:     wcss.append(kmeans.inertia_)
    ...:
    ...:

In [26]: plt.plot(range(1, 11), wcss, 'bx-')
    ...: plt.title('The Elbow Method')
    ...: plt.xlabel('Number of clusters')
    ...: plt.ylabel('WCSS')
    ...: plt.show()
```


The Elbow Method
WCSS
200
175
150
125
100
75
50
25
2
4
6
8
10
Number of clusters

Build clusters

```
In [27]: # Running kmeans to our optimal number of clusters

In [28]: kmeans = KMeans(n_clusters= 4)

In [29]: clusters = kmeans.fit_predict(X_scaled)

In [30]: clusters
Out[30]:
array([2, 1, 1, 2, 1, 1, 0, 0, 1, 2, 0, 3, 1, 0, 3, 0, 3, 2, 3, 1, 0, 1,
       3, 2, 1, 3, 3, 1, 3, 0, 1, 1, 2, 3, 0, 0, 0, 0, 0, 2, 3, 2, 1, 0,
       3, 0, 0, 3, 3, 0])
```

Cluster Membership

```
In [31]: # Naming Clusters 1-4 instead of 0-3 and adding to dataframe

In [32]: y_kmeans1 = clusters + 1

In [33]: cluster = list(y_kmeans1)

In [34]: df['cluster'] = cluster

In [35]: df.head()
Out[35]:
            Murder  Assault  UrbanPop  Rape  cluster
States
Alabama       13.2      236        58  21.2        3
Alaska        10.0      263        48  44.5        2
Arizona        8.1      294        80  31.0        2
Arkansas       8.8      190        50  19.5        3
California     9.0      276        91  40.6        2
```

Cluster-1

```
In [36]: # States and counts in different clusters

In [37]: df[df['cluster']==1]
Out[37]:
```

States	Murder	Assault	UrbanPop	Rape	cluster
Connecticut	3.3	110	77	11.1	1
Delaware	5.9	238	72	15.8	1
Hawaii	5.3	46	83	20.2	1
Indiana	7.2	113	65	21.0	1
Kansas	6.0	115	66	18.0	1
Massachusetts	4.4	149	85	16.3	1
New Jersey	7.4	159	89	18.8	1
Ohio	7.3	120	75	21.4	1
Oklahoma	6.6	151	68	20.0	1
Oregon	4.9	159	67	29.3	1
Pennsylvania	6.3	106	72	14.9	1
Rhode Island	3.4	174	87	8.3	1
Utah	3.2	120	80	22.9	1
Virginia	8.5	156	63	20.7	1
Washington	4.0	145	73	26.2	1
Wyoming	6.8	161	60	15.6	1

```
In [38]: len(df[df['cluster']==1])
Out[38]: 16
```

Cluster-2

```
In [39]: df[df['cluster']==2]
Out[39]:
            Murder  Assault  UrbanPop  Rape  cluster
States
Alaska        10.0      263        48  44.5        2
Arizona        8.1      294        80  31.0        2
California     9.0      276        91  40.6        2
Colorado       7.9      204        78  38.7        2
Florida       15.4      335        80  31.9        2
Illinois      10.4      249        83  24.0        2
Maryland      11.3      300        67  27.8        2
Michigan      12.1      255        74  35.1        2
Missouri       9.0      178        70  28.2        2
Nevada        12.2      252        81  46.0        2
New Mexico    11.4      285        70  32.1        2
New York      11.1      254        86  26.1        2
Texas         12.7      201        80  25.5        2

In [40]: len(df[df['cluster']==2])
Out[40]: 13
```

Cluster-3

```
In [41]: df[df['cluster']==3]
Out[41]:
```

States	Murder	Assault	UrbanPop	Rape	cluster
Alabama	13.2	236	58	21.2	3
Arkansas	8.8	190	50	19.5	3
Georgia	17.4	211	60	25.8	3
Louisiana	15.4	249	66	22.2	3
Mississippi	16.1	259	44	17.1	3
North Carolina	13.0	337	45	16.1	3
South Carolina	14.4	279	48	22.5	3
Tennessee	13.2	188	59	26.9	3

```
In [42]: len(df[df['cluster']==3])
Out[42]: 8
```

Cluster-4

```
In [43]: df[df['cluster']==4]
Out[43]:
```

States	Murder	Assault	UrbanPop	Rape	cluster
Idaho	2.6	120	54	14.2	4
Iowa	2.2	56	57	11.3	4
Kentucky	9.7	109	52	16.3	4
Maine	2.1	83	51	7.8	4
Minnesota	2.7	72	66	14.9	4
Montana	6.0	109	53	16.4	4
Nebraska	4.3	102	62	16.5	4
New Hampshire	2.1	57	56	9.5	4
North Dakota	0.8	45	44	7.3	4
South Dakota	3.8	86	45	12.8	4
Vermont	2.2	48	32	11.2	4
West Virginia	5.7	81	39	9.3	4
Wisconsin	2.6	53	66	10.8	4

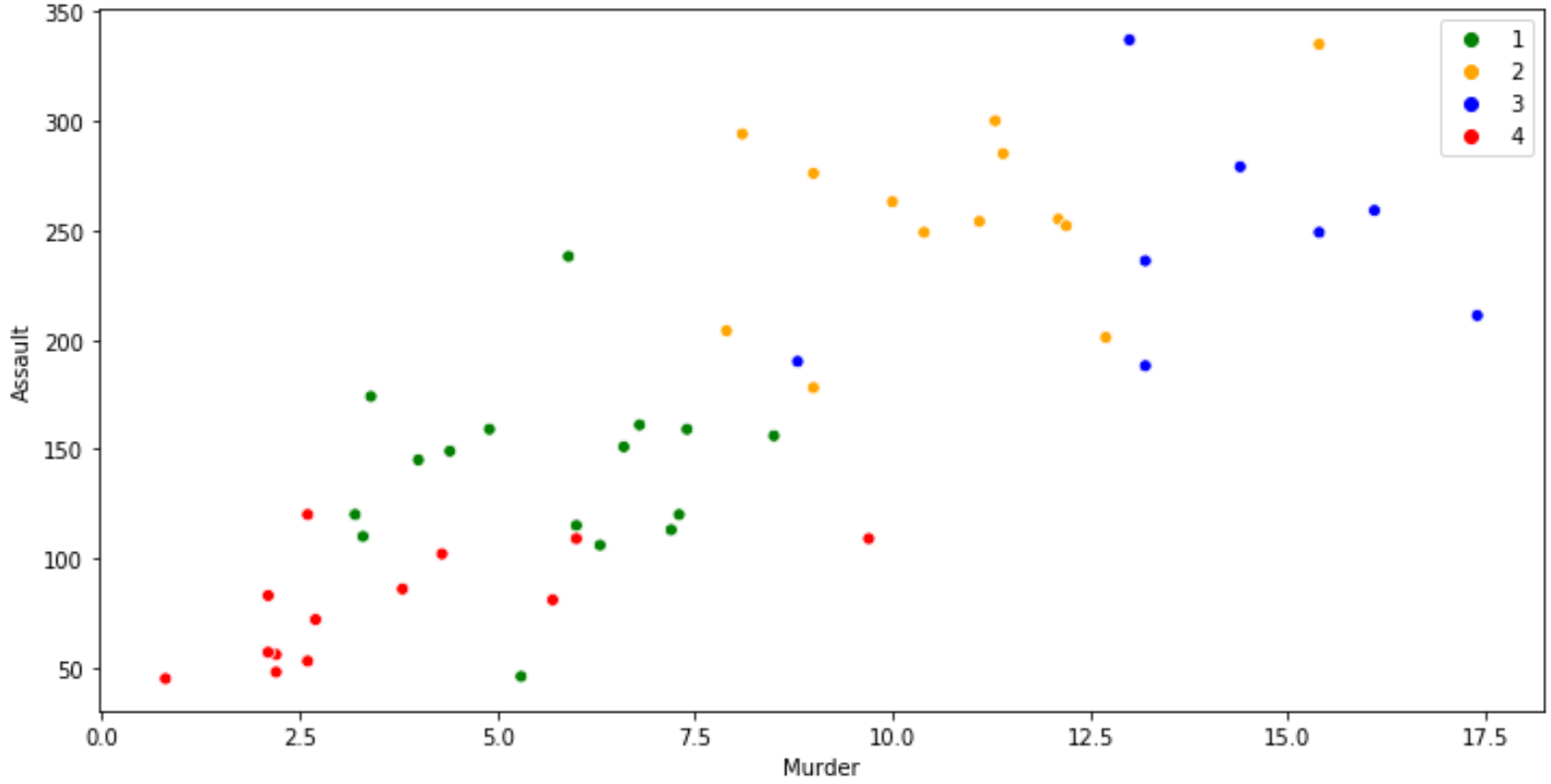
```
In [44]: len(df[df['cluster']==4])
Out[44]: 13
```

Cluster Profiling

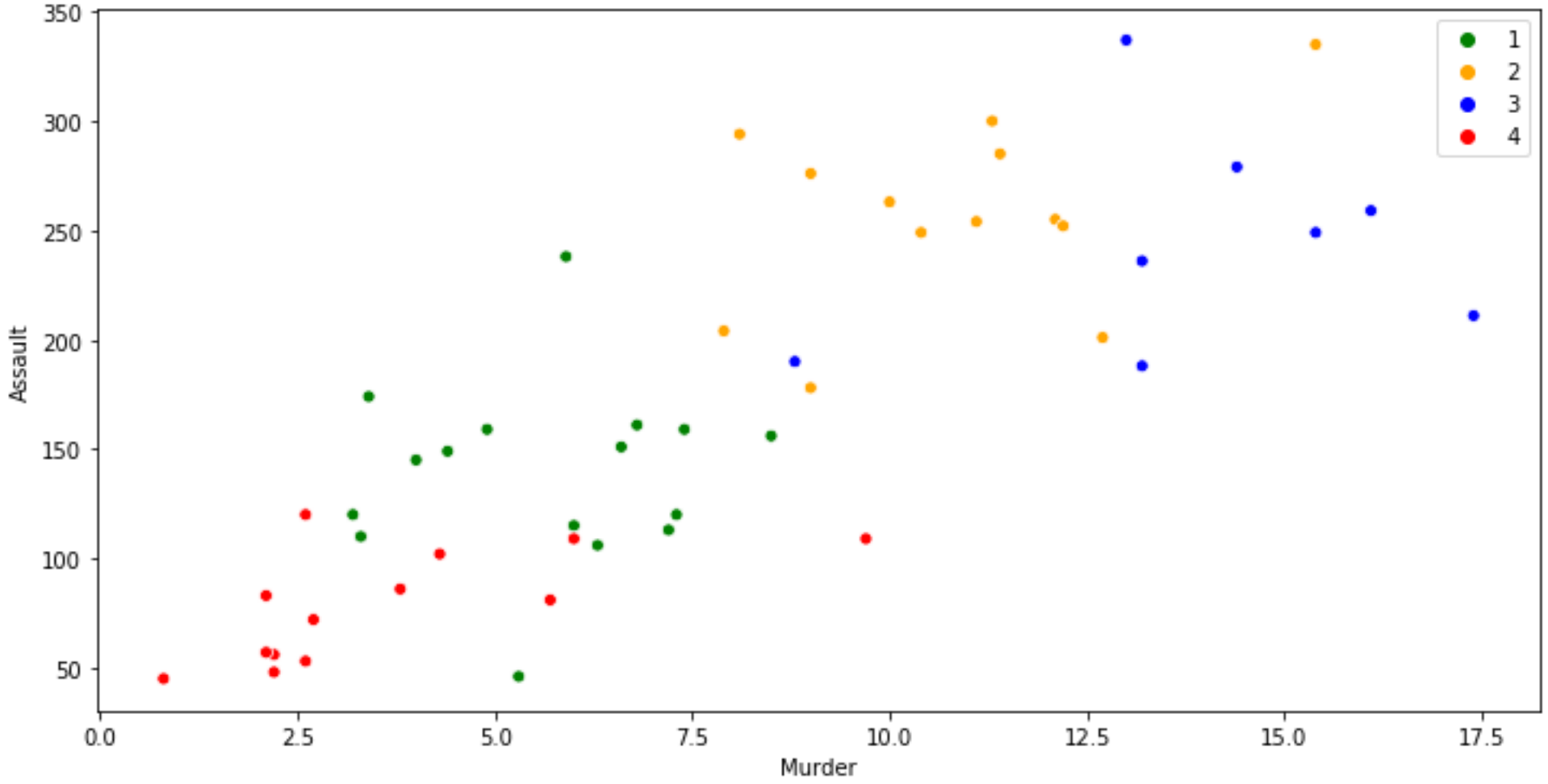
```
In [45]: # Mean of clusters 1 to 4

In [46]: kmeans_mean_cluster = pd.DataFrame(round(df.groupby('cluster').mean(),1))

In [47]: kmeans_mean_cluster
Out[47]:
         Murder  Assault  UrbanPop  Rape
cluster
1           5.7    138.9      73.9  18.8
2          10.8    257.4      76.0  33.2
3          13.9    243.6      53.8  21.4
4           3.6     78.5      52.1  12.2
```

Plot

```
In [48]: x = df['Murder']
    ...: y = df['Assault']
    ...: plt.figure(figsize=(12,6))
    ...: sns.scatterplot(x, y, hue=y_kmeans1,
    ...:     palette=['green','orange','blue','red'], legend='full')
    ...:
Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x23393fb3710>
```



Plot


Assault
Murder
1
2
3
4
350
300
250
200
150
100
50
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Cophenetic Correlation

Average Linkage

	A	B	C	D	E
A	*	6.32 [7.49]	7.07 [7.49]	1.41 [1.41	7.16 [7.49]
B		*	1.41 [1.74]	7.62 [7.49]	1.12 [1.12]
C			*	8.25 [7.49]	2.06 [1.74]
D				*	8.50 [7.49]
E					*

$We\ merged\ b\ \&\ e\ with\ (b,e)\ at\ a\ distance\ 1.12$

$Then\ we\ merged\ a\ \&\ d\ with\ (a,d)\ at\ a\ distance\ 1.41$

$We\ merged\ c\ with\ ((b,e),c)\ at\ a\ distance\ 1.74$

$Then\ we\ merged\ (a,d)\ with\ ((b,e),c)\ at\ a\ distance\ 7.49$

Average Linkage	
Dist	Merged at dist
6.32	7.49
7.07	7.49
1.41	1.41
7.16	7.49
1.41	1.74
7.62	7.49
1.12	1.12
8.25	7.49
2.06	1.74
8.5	7.49
Correl =	0.980423166

Silhouette Analysis

Silhouette analysis can be used to determine the degree of separation between clusters. For each sample:

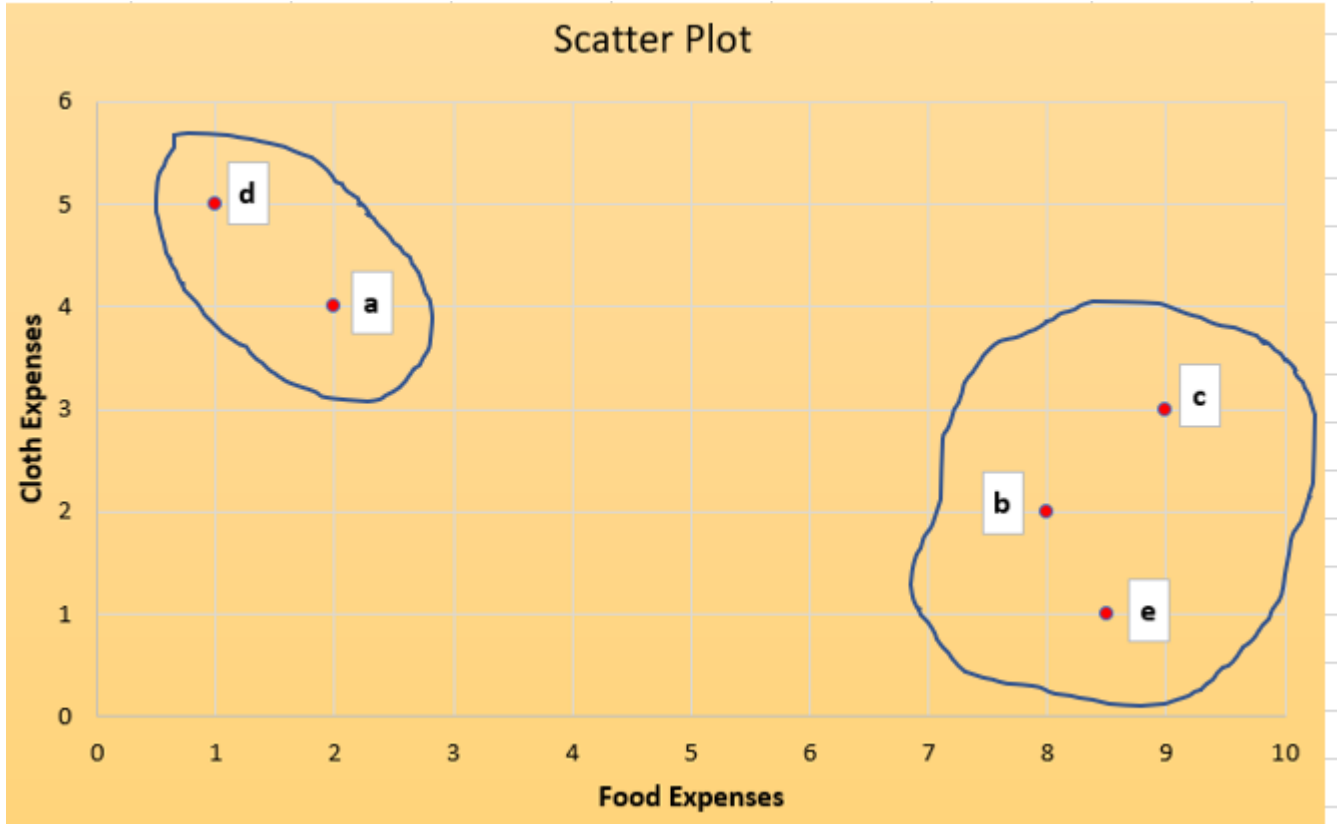
- Compute the average distance from all data points in the same cluster (ai).
- Compute the average distance from all data points in the closest cluster (bi).
- Compute the coefficient:

$$\frac{b^i - a^i}{max(a^i, b^i)}$$

The coefficient can take values in the interval [-1, 1].

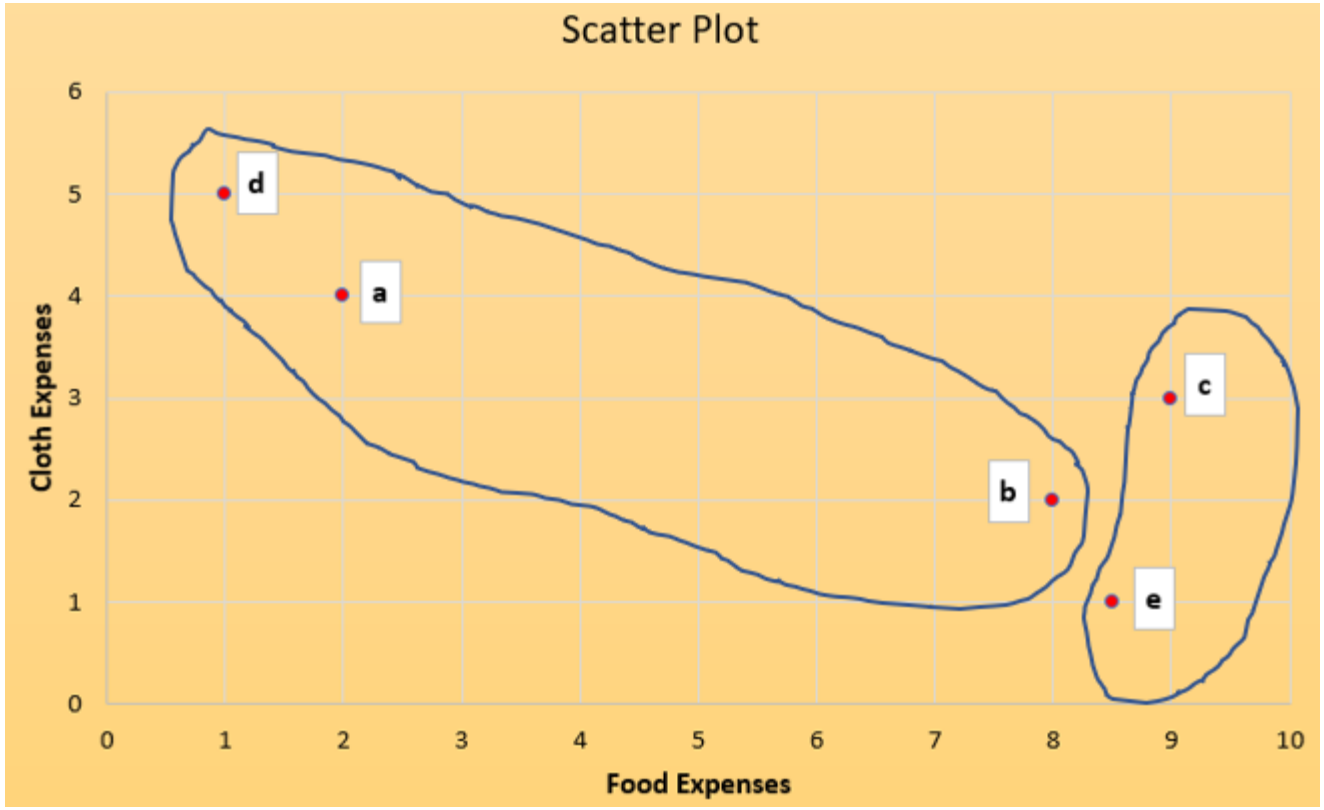
- If it is 0 –> the sample is very close to the neighbouring clusters.
- If it is 1 –> the sample is far away from the neighbouring clusters.
- If it is -1 –> the sample is assigned to the wrong clusters.

Silhoutte_score (K Means)



	Silhoutte Score for a		
		d	
C1	ad	1.41	A
C2	ab	6.32	
C2	ac	7.07	
C2	ae	7.16	
	AVG =	6.85	B
	Sil_Sc(a)=	0.794161	
	As it is close to 1,		
	righty allotted C1		

Silhoutte_score (K Means)

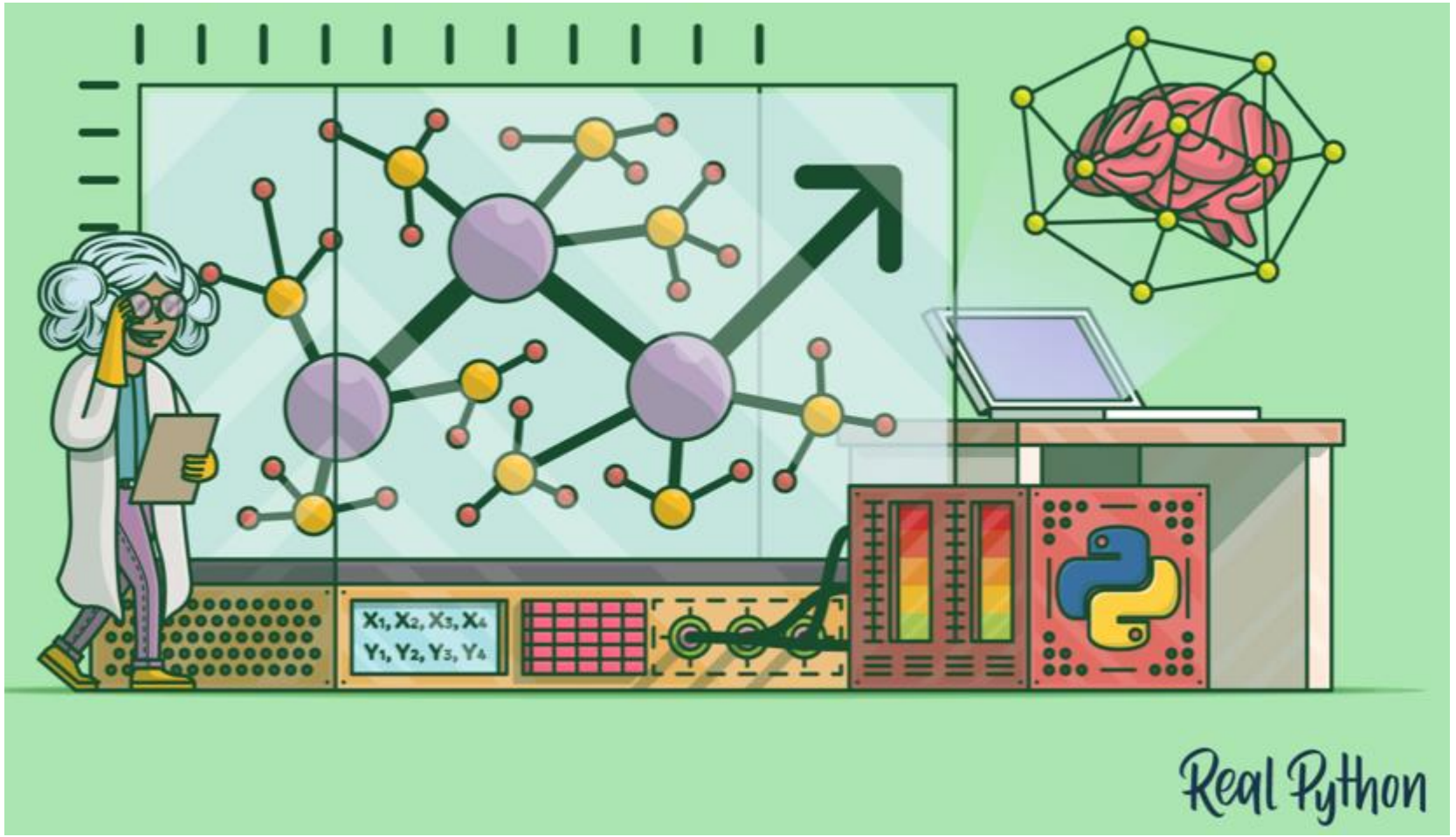


Silhoutte Score for b			
		d	
C1	ba	6.32	
C1	bd	7.62	
	AVG=	6.97	A
C2	be	1.12	
C2	bc	1.41	
	AVG=	1.265	B
	Sil_Sc(b)=	-0.81851	
	As it is close to -1,		
	wrongly allotted to C1		

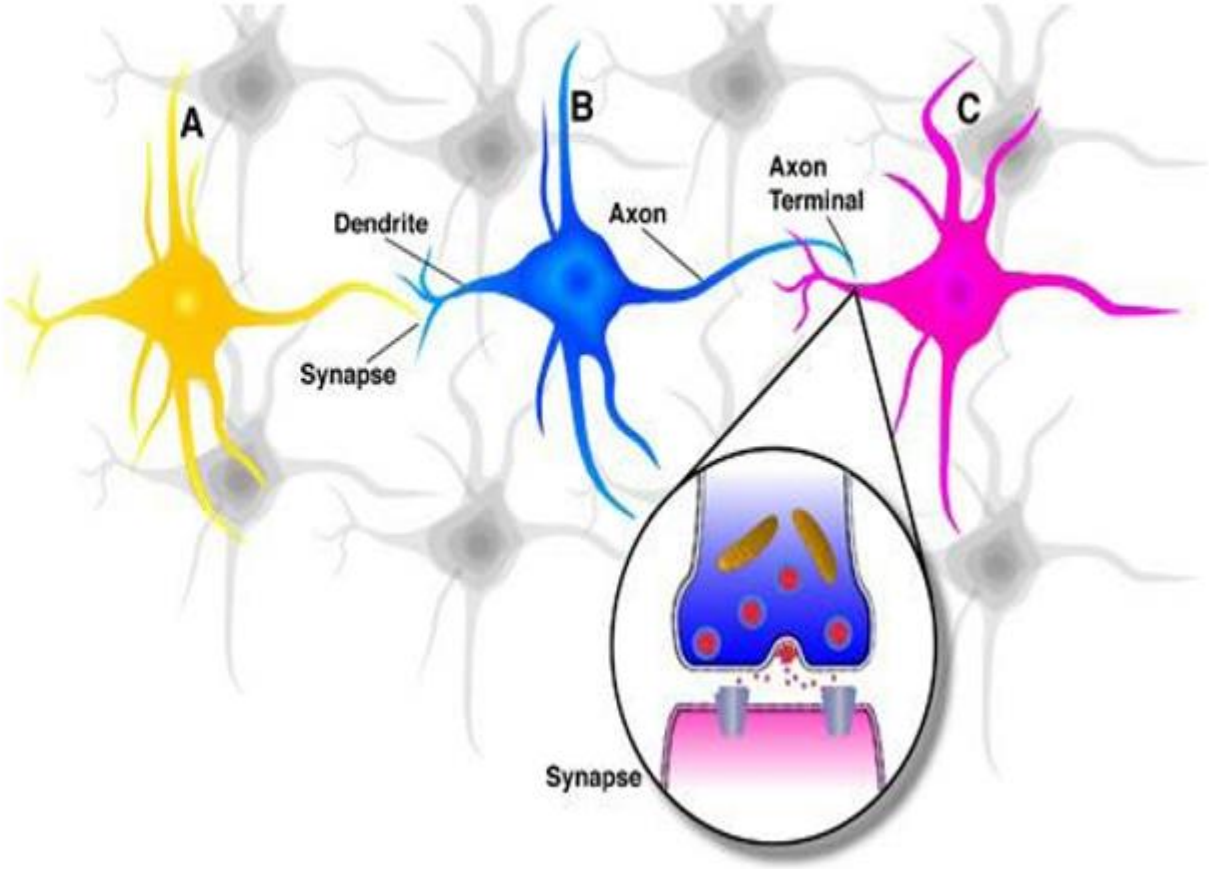
Artificial Neural Networks

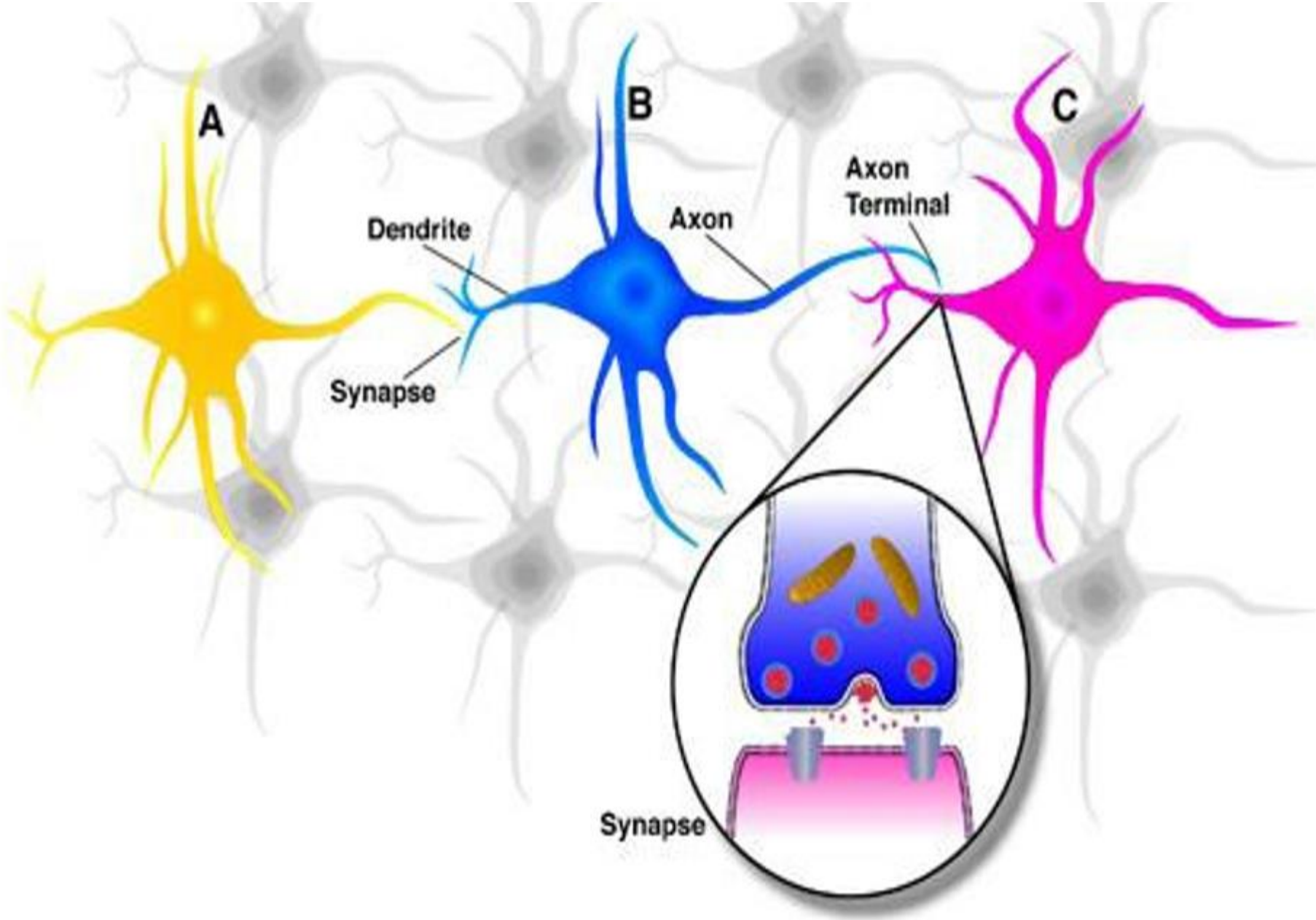
Human Brain!

Why Brain!

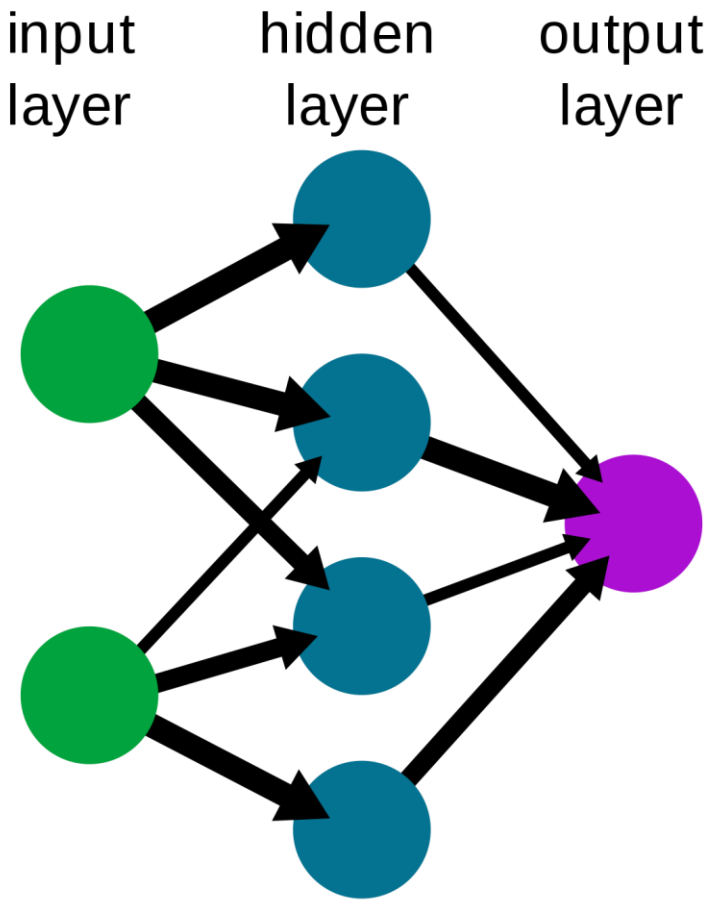


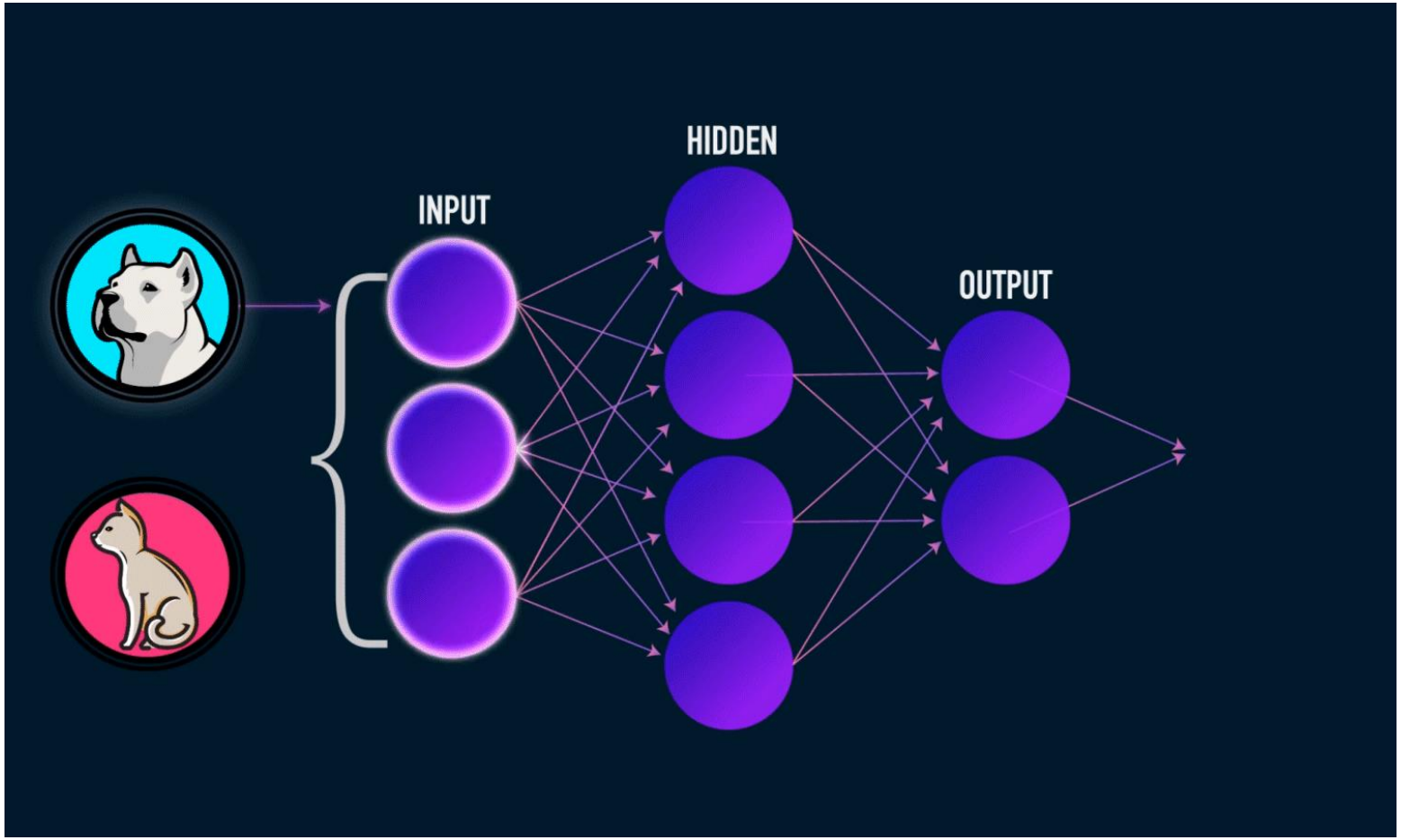
Biological Neural Network



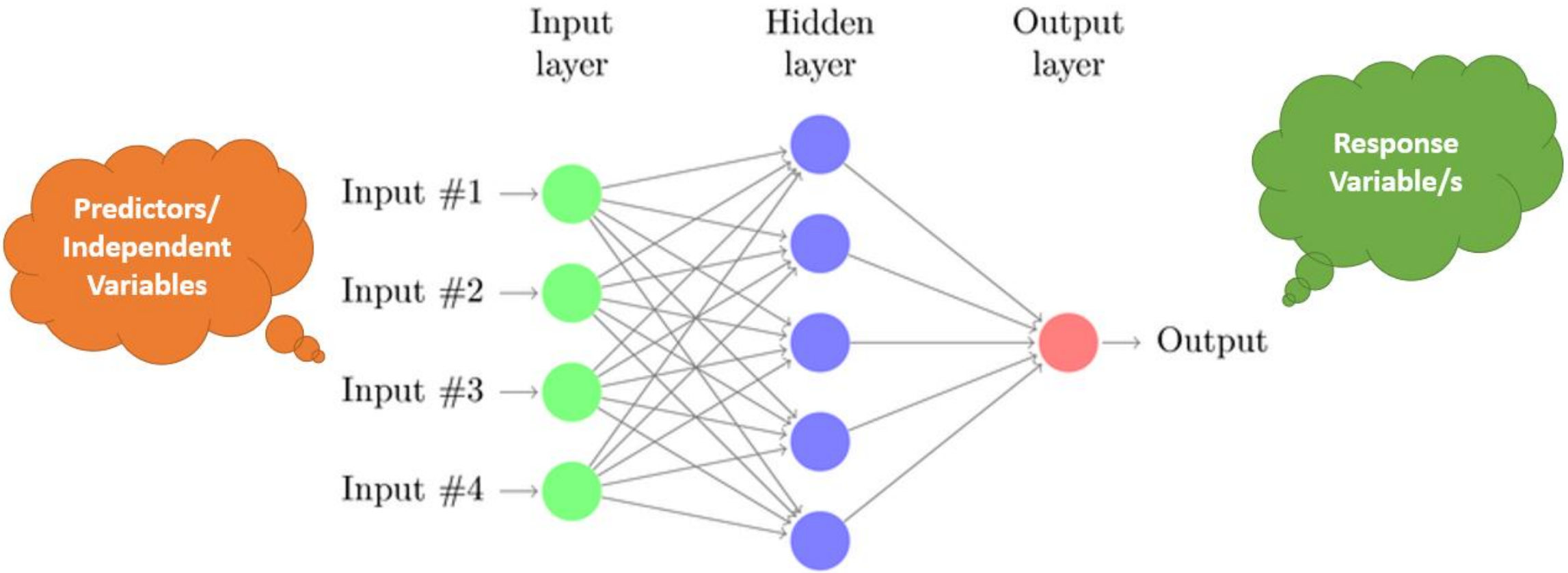

A
B
C
Dendrite
Axon
Axon Terminal
Synapse
Synapse

A simple neural network



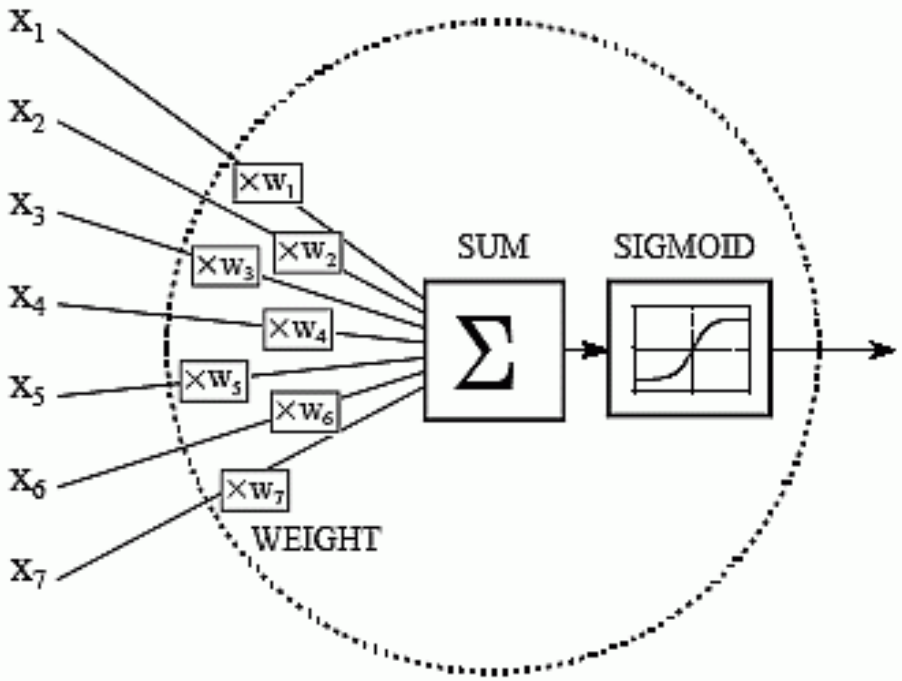

HIDDEN
INPUT
OUTPUT

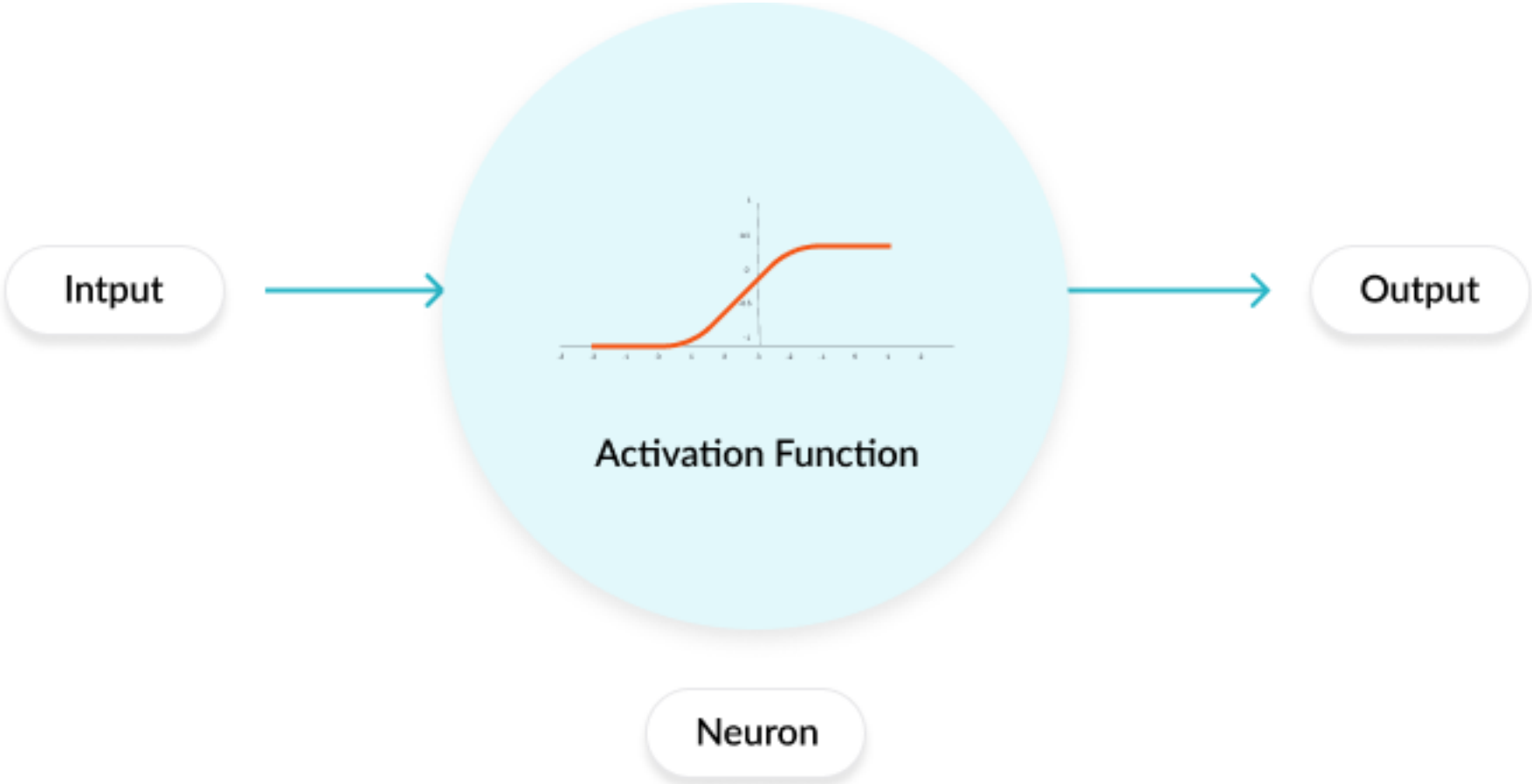
Network


Input layer
Hidden layer
Output layer
Predictors/ Independent Variables
Input #1
Input #2
Input #3
Input #4
Output
Response Variable/s

Activation Functions

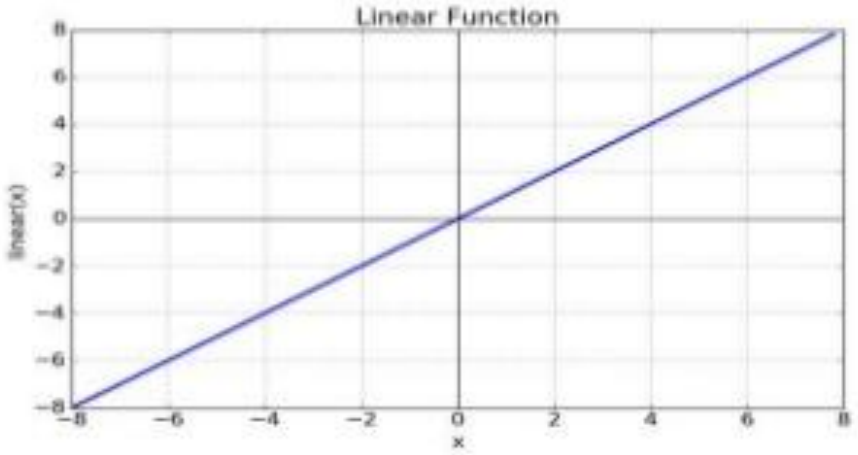
FIGURE 26-6
Neural network active node. This is a flow diagram of the active nodes used in the hidden and output layers of the neural network. Each input is multiplied by a weight (the w_N values), and then summed. This produces a single value that is passed through an "s" shaped nonlinear function called a *sigmoid*. The sigmoid function is shown in more detail in Fig. 26-7.



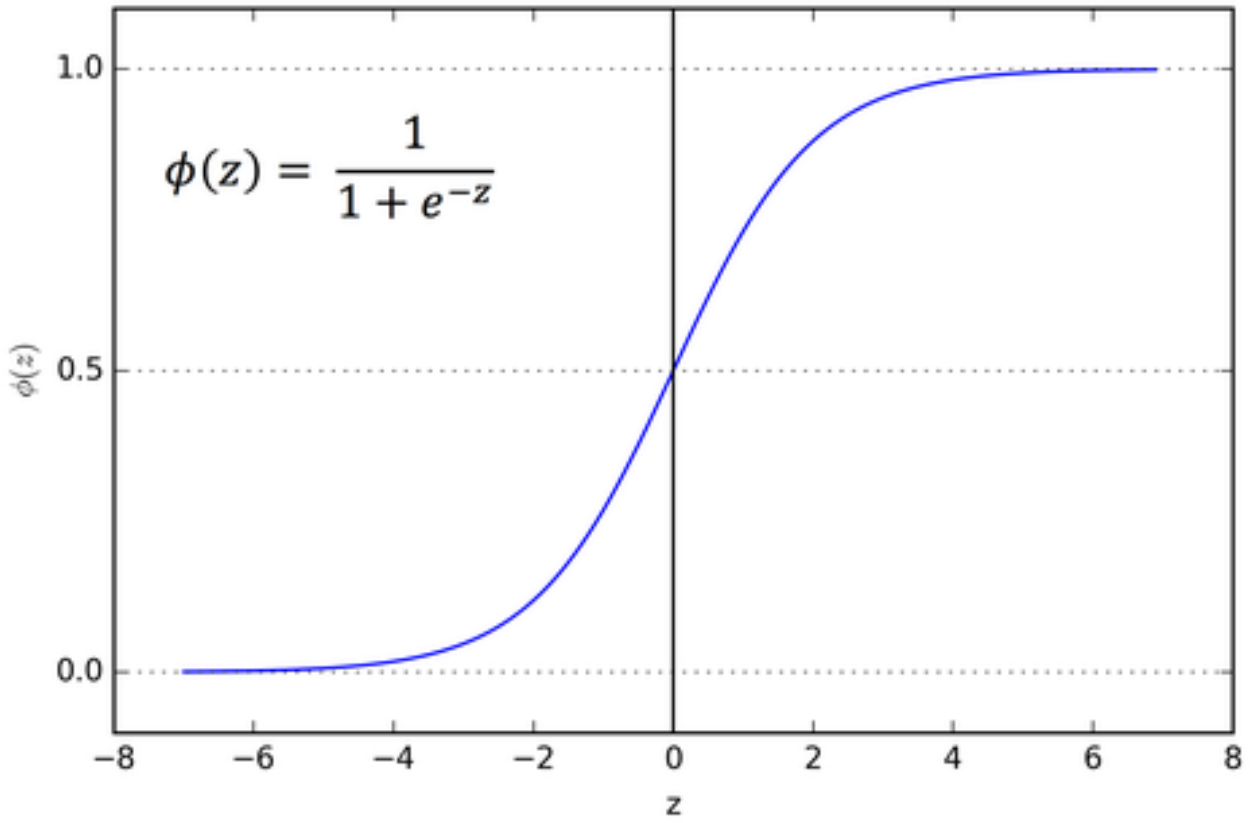

Intput
Activation Function
Output
Neuron

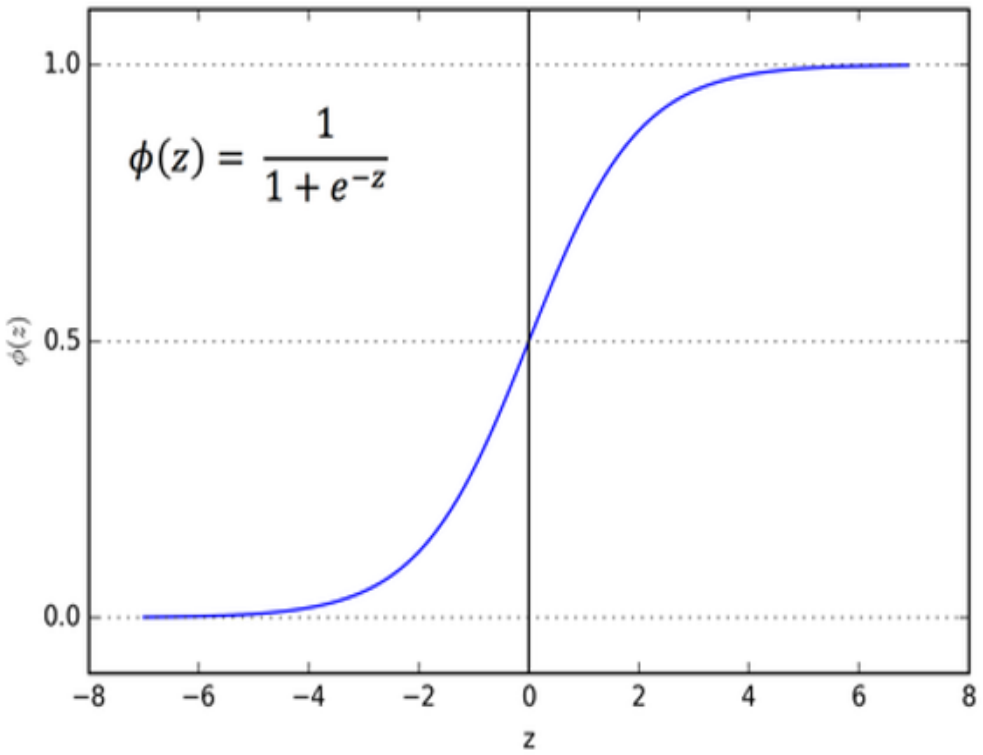
Linear or Identity Activation Function

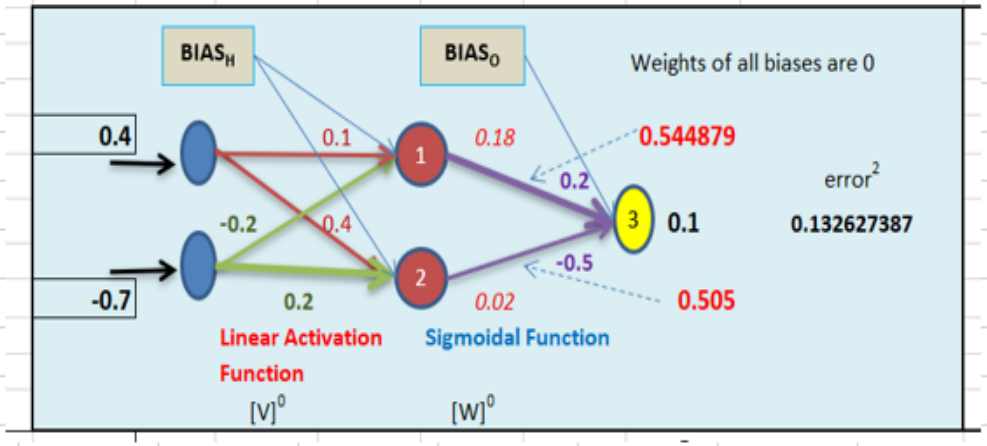
- As you can see the function is a line or linear. Therefore, the output of the functions will not be confined between any range.
- **Equation :** f(x) = x
- **Range :** (-infinity to infinity)
- It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.



Sigmoid or logistic







At node 1

First input(0.4) multiplied by 0.1 + Second input (-0.7) multiplied by -0.2 **0.18**

but, 0.18 will not be taken forward!

Sigmoid function will translate 0.18 to **0.54488**

For node 1,	*0.18*	**-0.18**
For node 2,	*0.02*	**-0.02**

for using here

Step 4 Find output from hidden which will be input for output node.

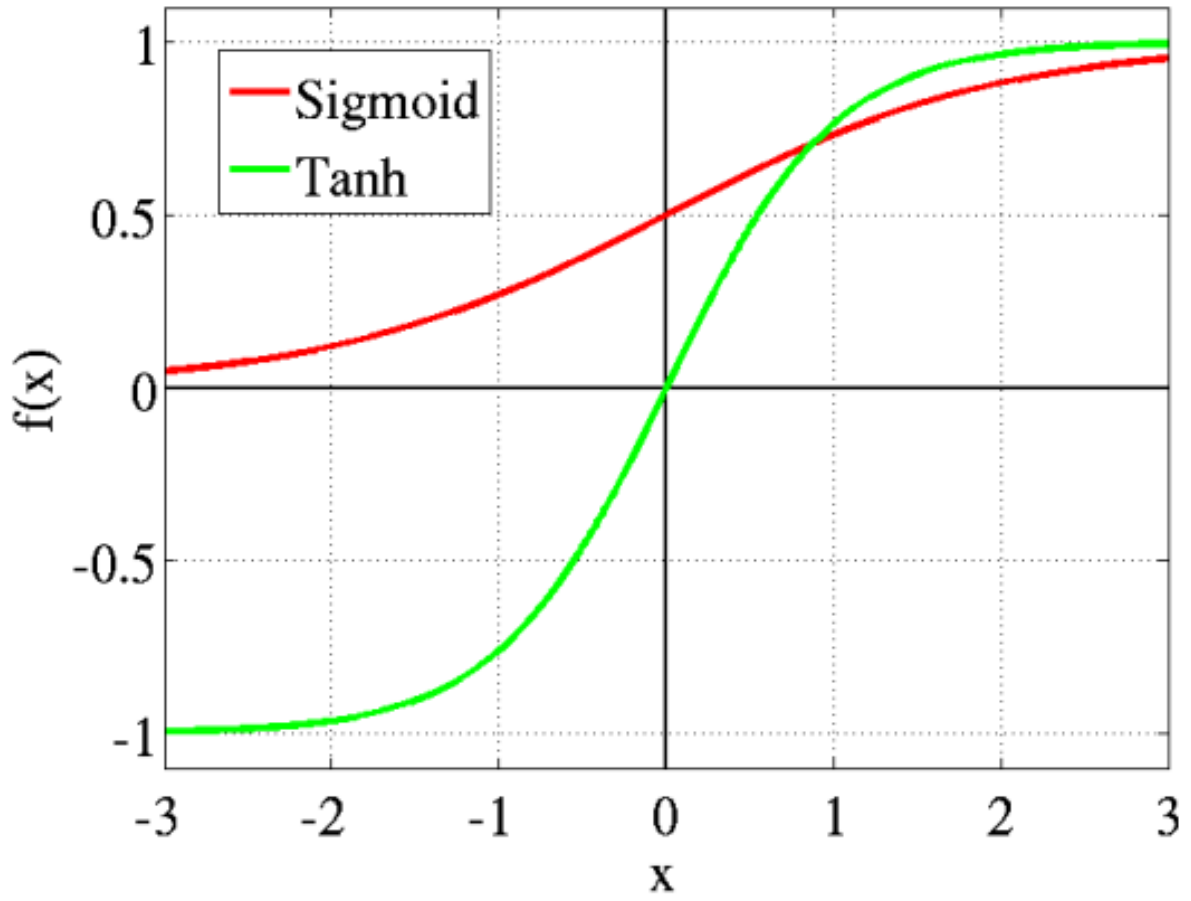
Sigmoidal Function→ 1/(1+e-0.18) **0.544879**

1/(1+e-0.02) **0.505**

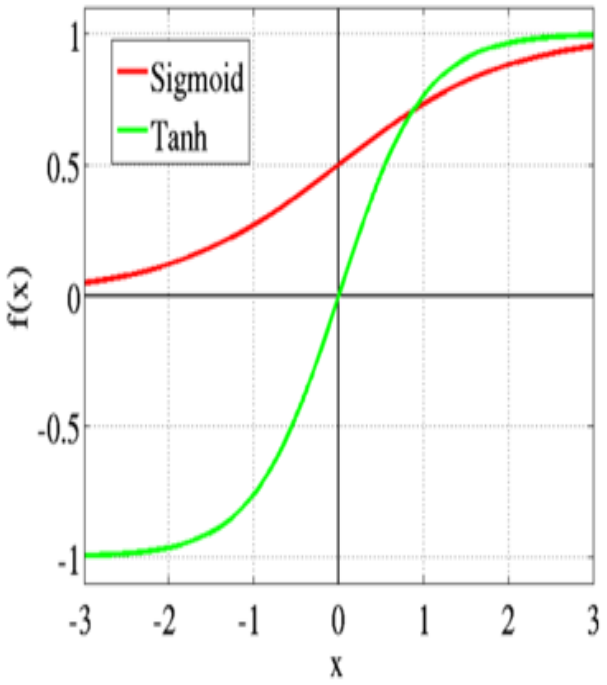
carefully note, output from hidden nodes is just not 0.18 and 0.02

rather these are converted by sigmoidal function!

tanh



tanh

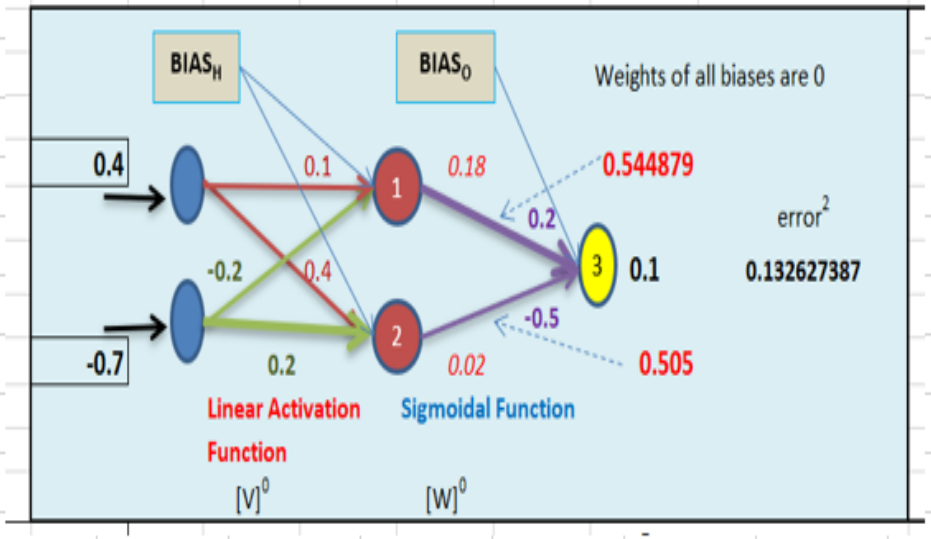


TanH

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

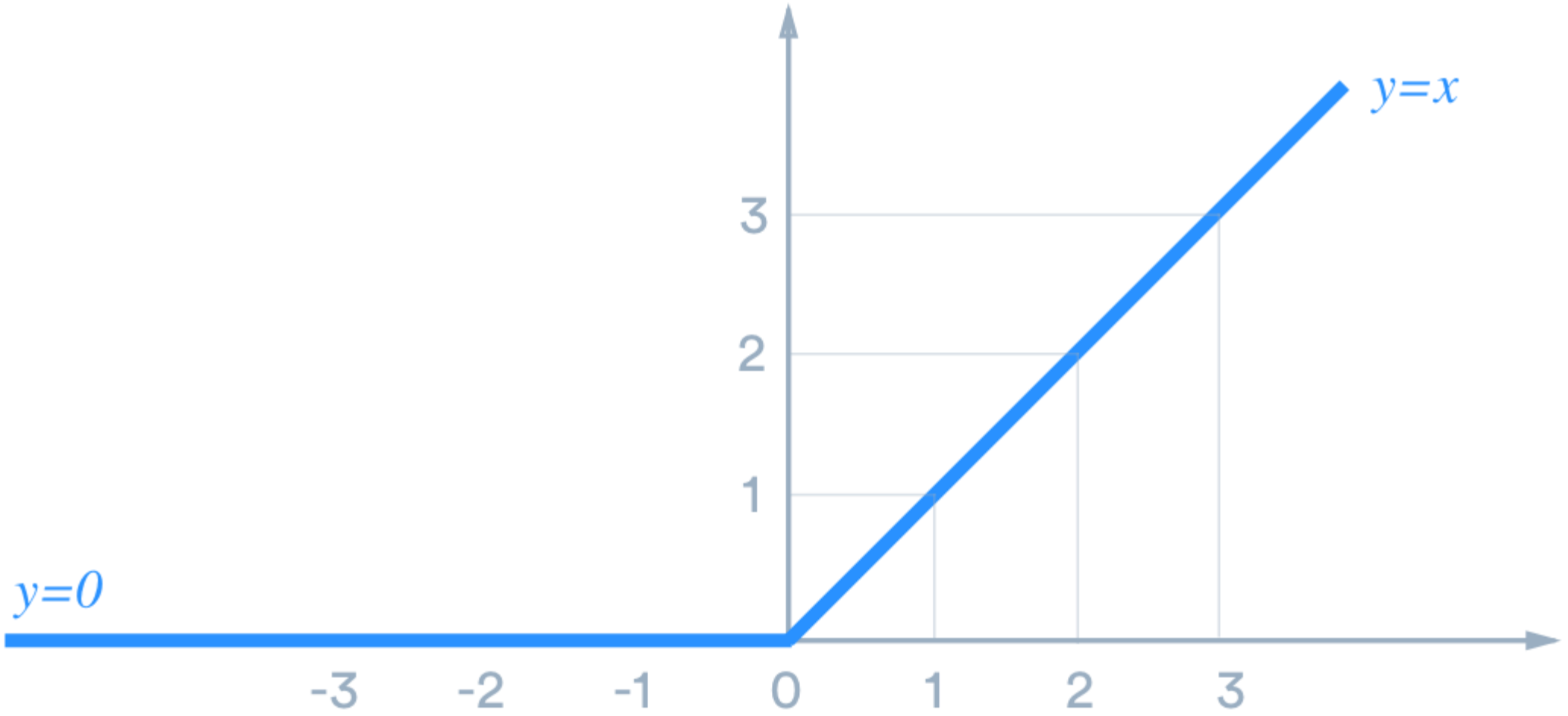
At node 1

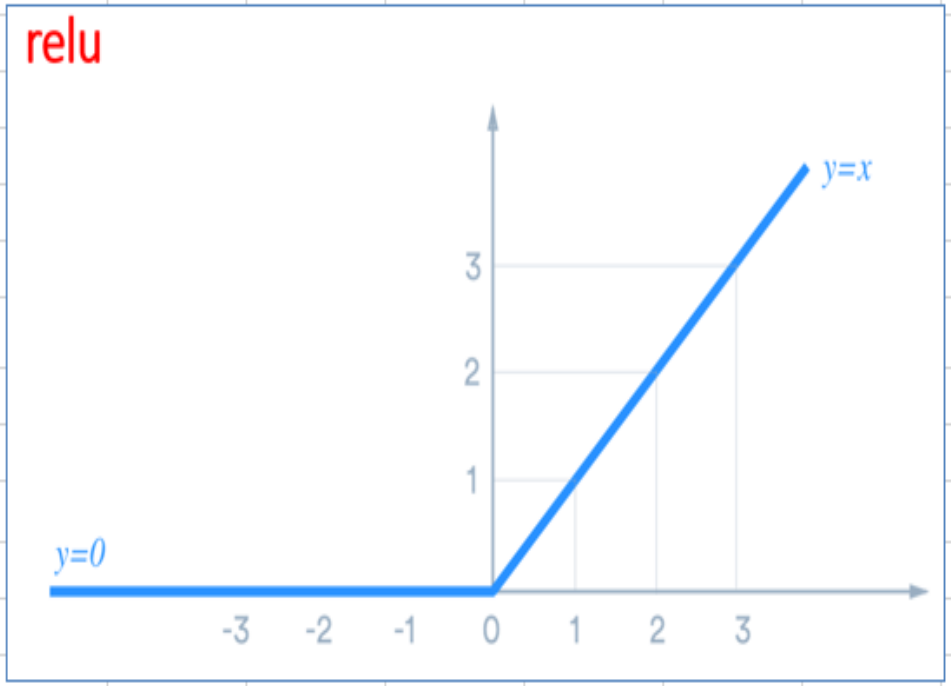
First input(0.4) multiplied by 0.1 + Second input (-0.7) multiplied by -0.2		0.18
but, 0.18 will not be taken forward!		
tanH function will translate 0.18 to	0.17808	0.178081



Denom	1.697676
2/Denom	1.178081
(2/Denom)-1	0.178081

relu

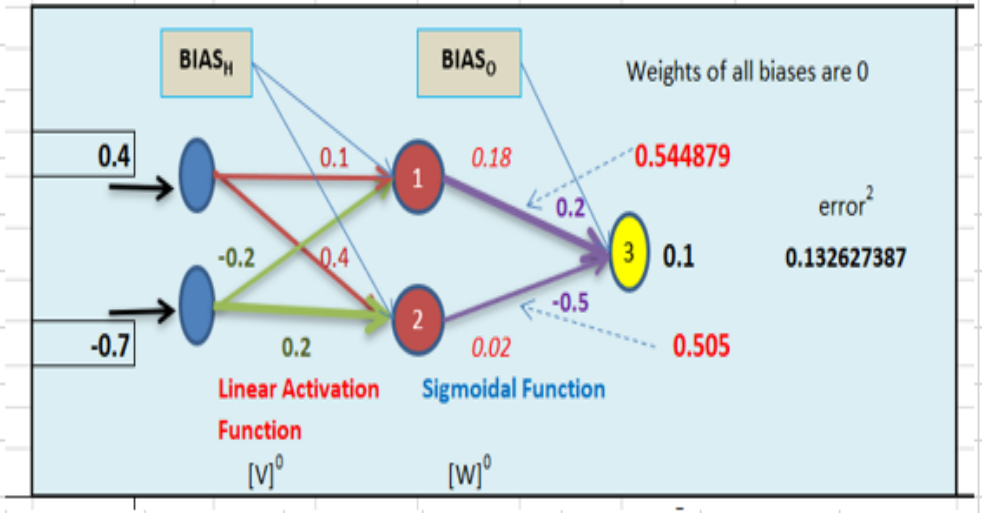




Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$

At node 1

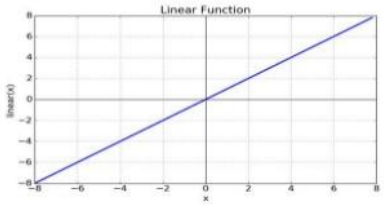
First input(0.4) multiplied by 0.1 + Second input (-0.7) multiplied by -0.2	0.18
but, 0.18 will be subjected to conditions as mentioned above, as 0.18 is >=0	
relU function will translate 0.18 to	**0.18**

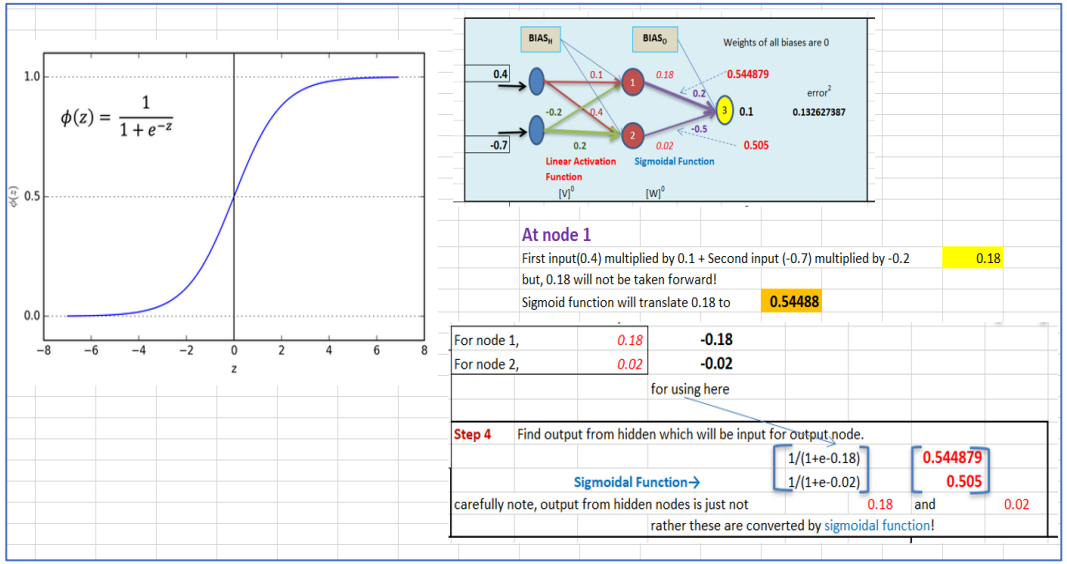


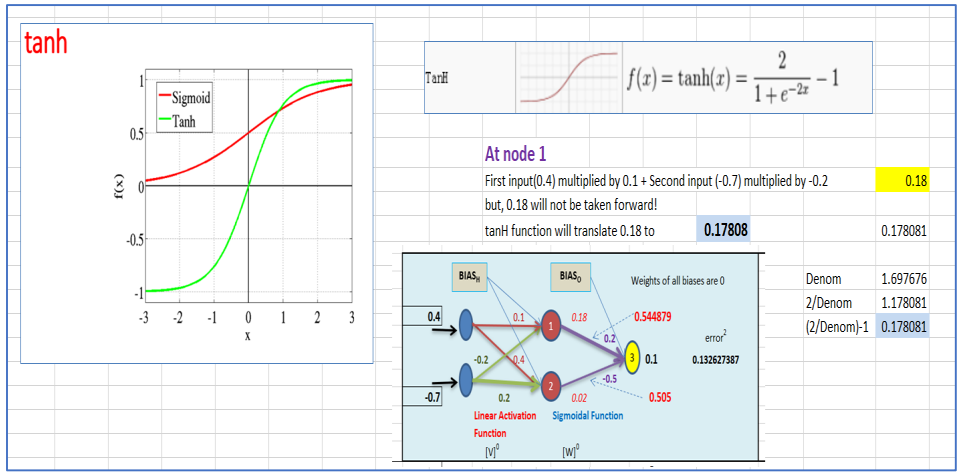
Lets see all together!

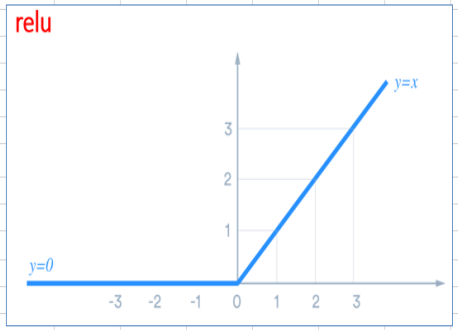
Linear or Identity Activation Function

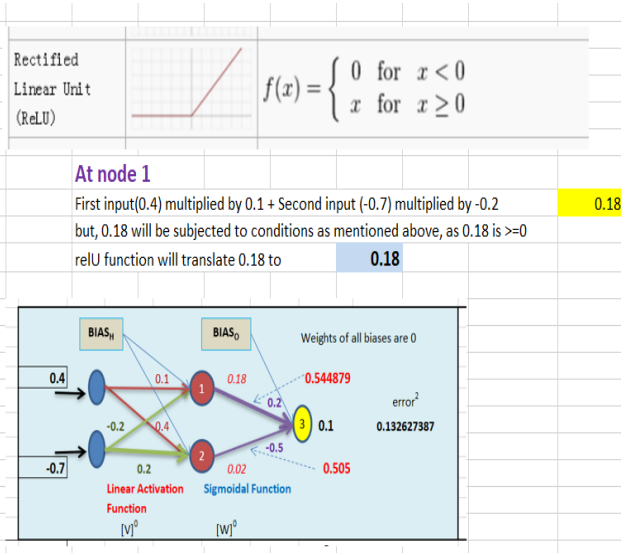
- As you can see the function is a line or linear. Therefore, the output of the functions will not be confined between any range.
- **Equation :** f(x) = x
- **Range :** (-infinity to infinity)
- It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.



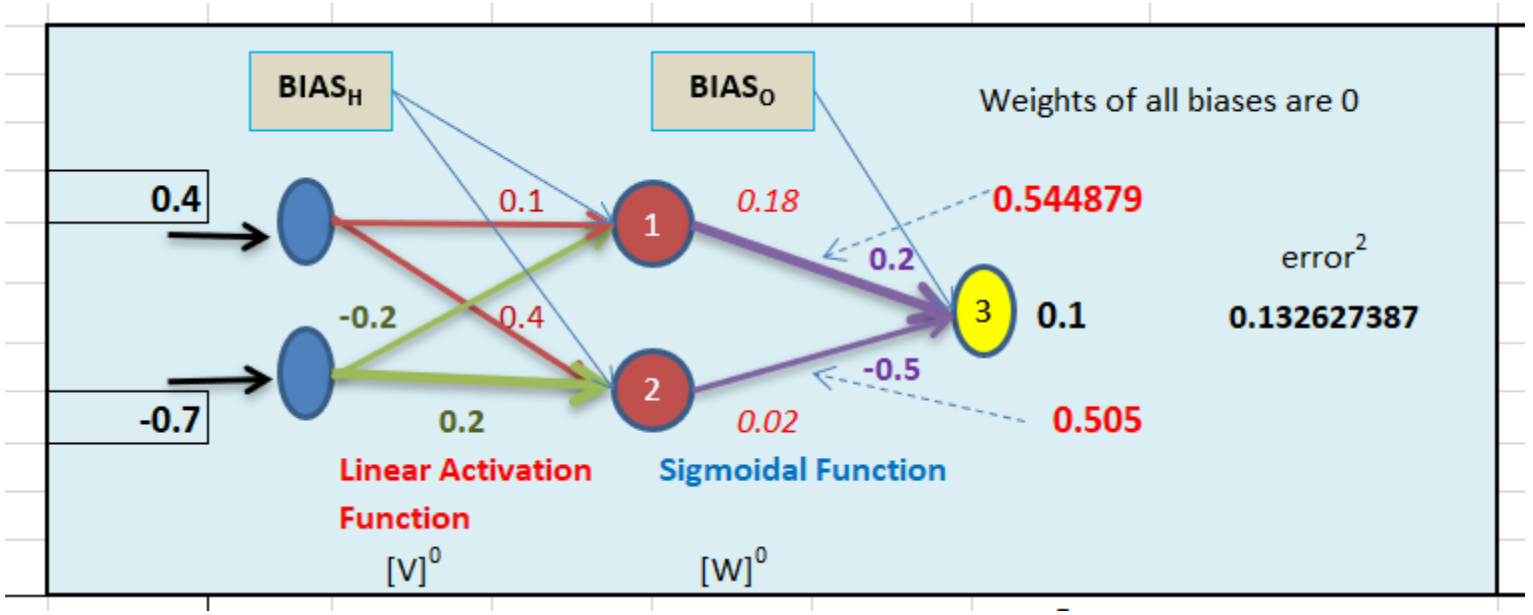








Gradient Descent



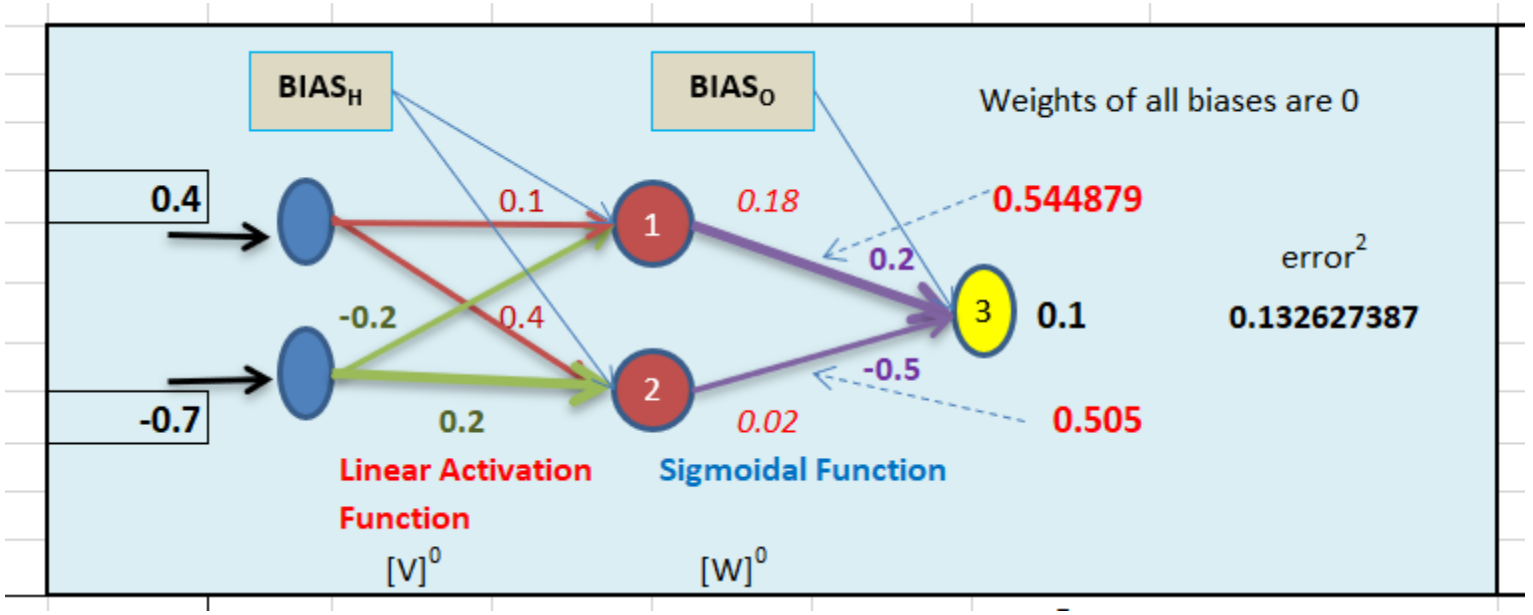
Gradient Descent

Sr No	I1	I2	O
1	**0.4**	**-0.7**	**0.1**
2	0.3	-0.5	0.05
3	0.6	0.1	0.3
4	0.2	0.4	0.25
5	0.1	-0.2	0.12

$[O]_I=\{I\}_I =$ output of input layer

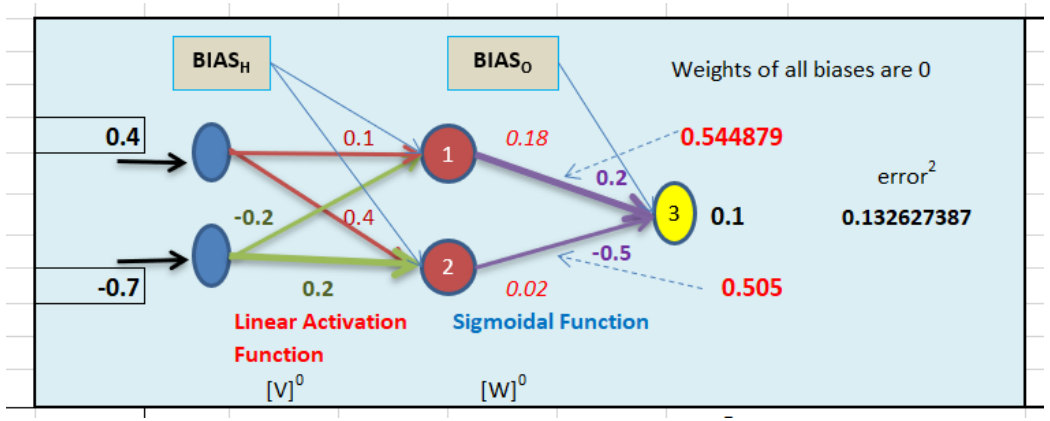
$$\begin{bmatrix} 0.4 \\ \\ -0.7 \end{bmatrix}$$

Gradient Descent



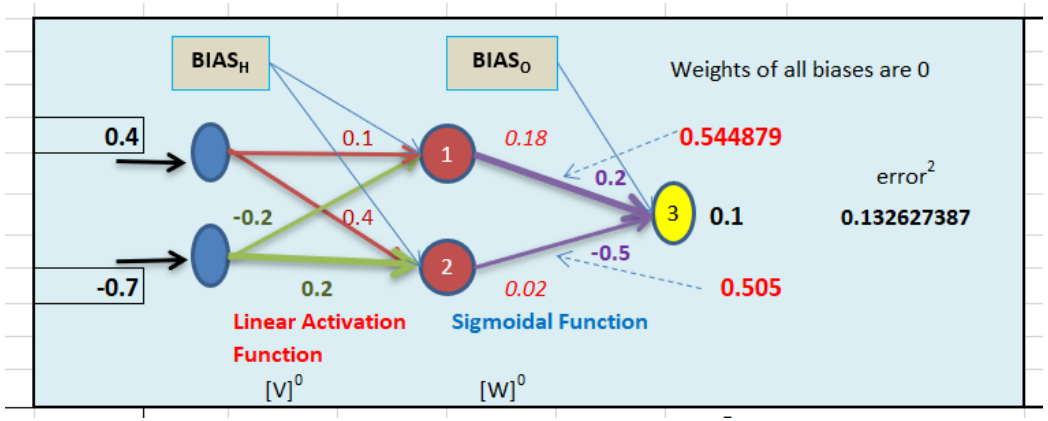
Gradient Descent

Step 1	All inputs are between -1 & +1 so no need to normalise.
We will take only 1st set I1 as 0.4 and I2 as =-0.7 ;0.4 & 0.7 are not autmtc	



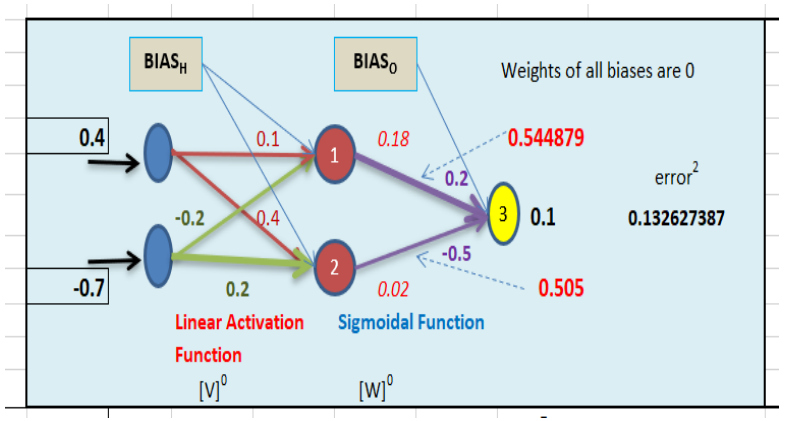
Gradient Descent

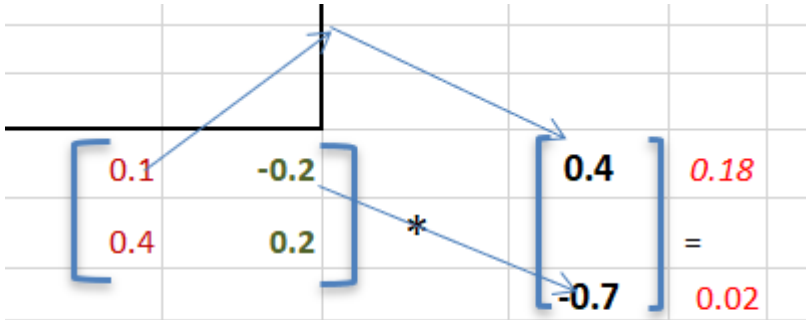
Step 2	Initialize wts as							
		$[V]^0$	0.1	0.4		$[W]^0$	0.2	
			-0.2	0.2			-0.5	
					2*2			2*1
					rows*columns			



Gradient Descent

Step 3	Find $\{I\}_H$	$= [V]^T * \{I\}_I$ or
	I stands for input, to Hidden layer.	
	T stands for transpose.	





Gradient Descent

For node 1,	*0.18*	**-0.18**
For node 2,	*0.02*	**-0.02**

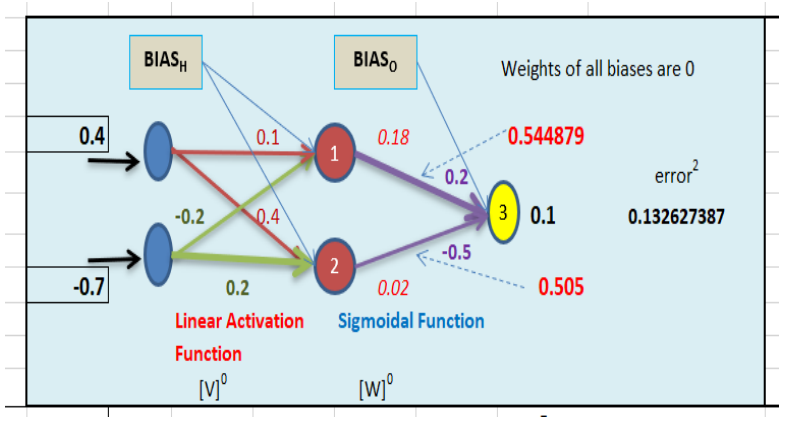
for using here

Step 4 Find output from hidden which will be input for output node.

Sigmoidal Function→	1/(1+e-0.18)	**0.544879**
	1/(1+e-0.02)	**0.505**

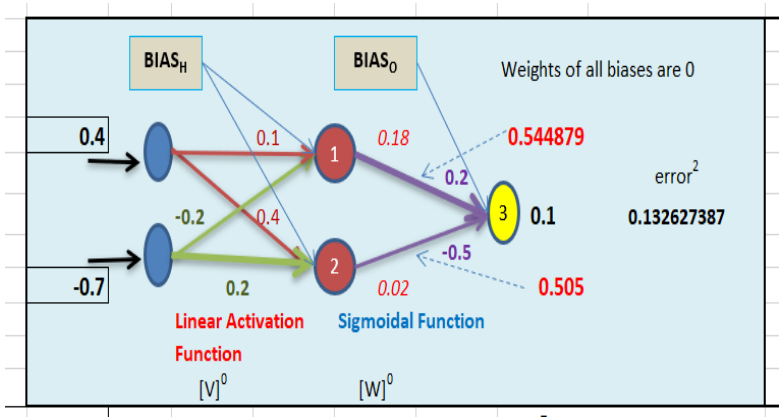
carefully note, output from hidden nodes is just not 0.18 and 0.02

rather these are converted by sigmoidal function!



Gradient Descent

Step 5	Calculate output of output layer/node again by Simple matrix application.								
	$[I]_O = [W]^T * \{O\}_H = ($	0.2	-0.5	) Mutiply	0.544879 0.505	=	-0.14352	And apply **sigmoidal function** and find output of output node/layer	0.143524
							For using minus sign before fitting in sigmoidal function =		



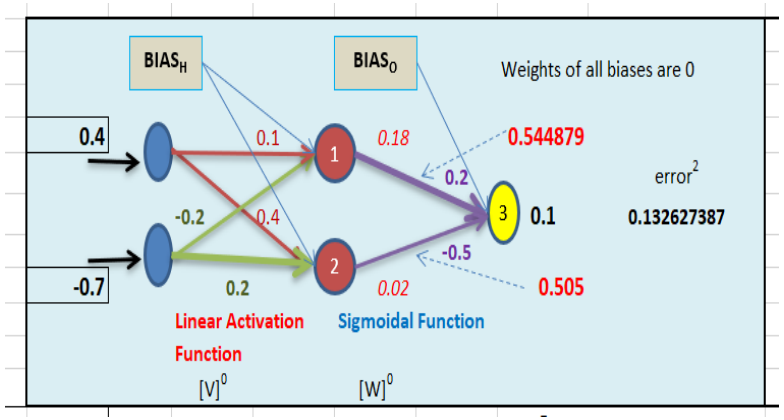
Gradient Descent

Step 6

1/[1+e-(-0.14352)]

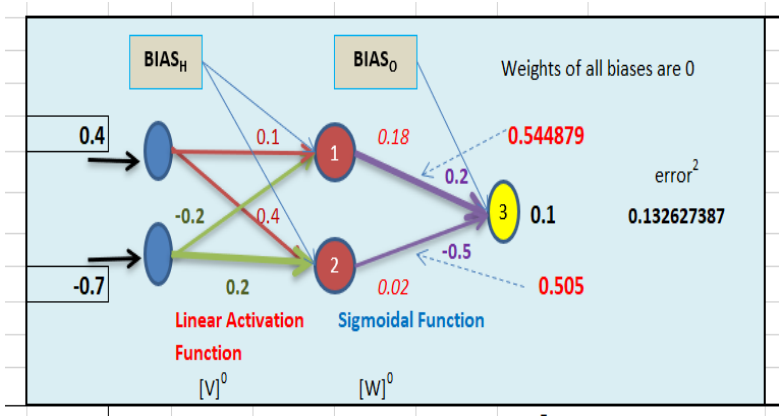
0.46418

This is answer! But erroneus!!



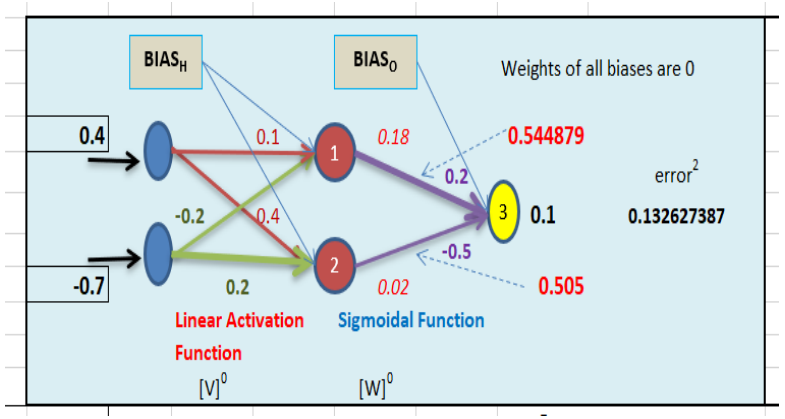
Gradient Descent

Step 7	Error = $(T_o - O_O)^2$ =		**0.132627**
	T_O=	0.1	
	O_O=	0.46418	0.13262739



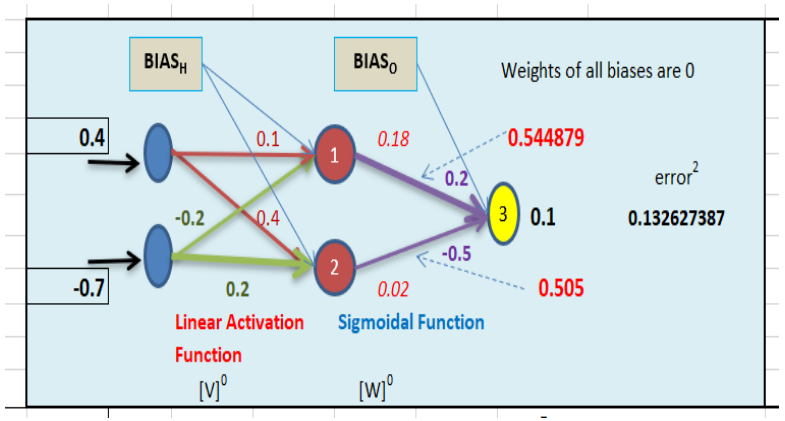
Gradient Descent

Step 8	Weight Adjustment		
	First find '**d**' as =	$(T_O-O_{OI})(O_{OI}(1-O_{Oi})$	-0.0906
	O_{OI} =Output of output layer/node for 1st case.		
	[Y]=[O]H<d> =	0.544879 0.505	< -0.09058 > = **-0.049354** **-0.045742**



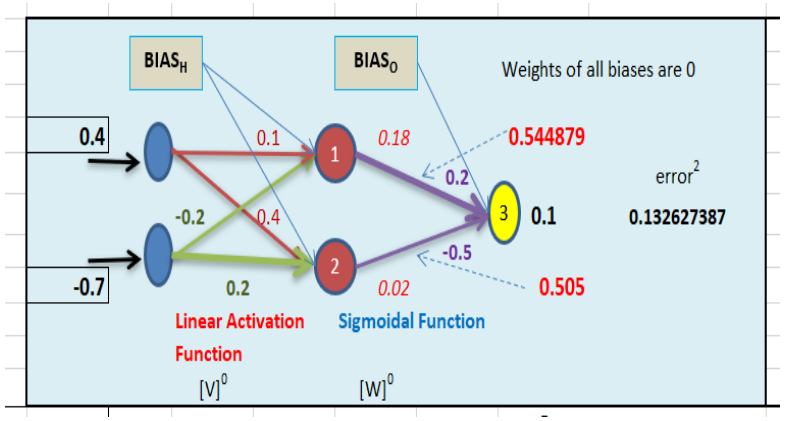
Gradient Descent

Step 9	Find $[\Delta W]^1$, as = $\alpha[\Delta W]^O + \eta[Y]$	
	Assume η as	0.6
	And initially, α as	0
	α is Momentum coefficient	
	η is Learning Rate	
	So 1st term will be 0 as α is 0.	
$[\Delta W]^1$ =	-0.049354 -0.045742	* 0.6
$[\Delta W]^1$ =	-0.029612 -0.027445	



Gradient Descent

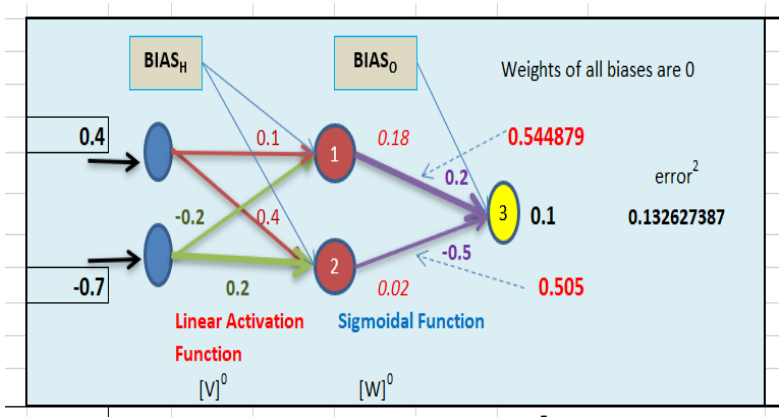
Step 10	**[e]**=[W]*{d}	0.2 -0.5	*	**-0.090577849**	=	-0.018 0.0453



Gradient Descent

Step 11	[d*] =	e, step 10's matrix*step 4's matrix*(1-step 4's matrix)

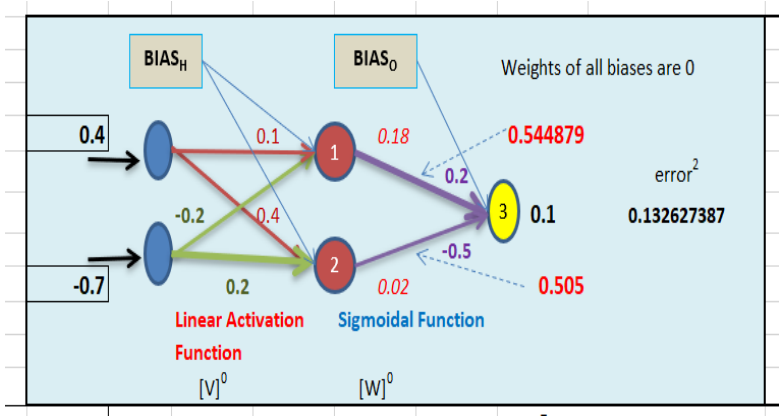
$$\begin{Bmatrix} -0.00449241 \\ 0.0113211 \end{Bmatrix}$$



Gradient Descent

Step 12 $[X] = \{O\}_I \langle d^* \rangle$

$$\begin{Bmatrix} 0.4 \\ -0.7 \end{Bmatrix} < -0.004492406 \quad 0.011321 > \begin{Bmatrix} -0.0018 & 0.004528 \\ 0.003145 & -0.00792 \end{Bmatrix}$$



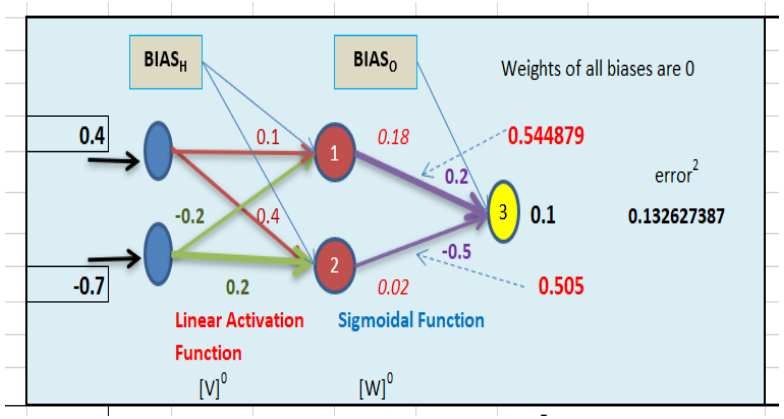
Gradient Descent

Step 13 Find $[\Delta V]^1$, as = $\alpha[\Delta V]^0 + \eta[X]$

Assume η as 0.6, AND α as zero.

$[\Delta V]^1$ is change in wts in Input Layer

0.6 *	-0.0018	0.004528	=	-0.00108	0.002717064
	0.003145	-0.00792		0.001887	-0.004754862

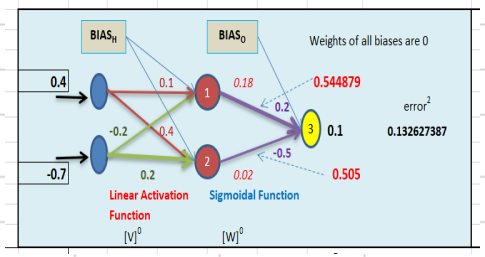


Gradient Descent

Step 14 New wts in Input Layer to Hidden Layer

$[V]^1$ = Original wts + required/estimated change in wts in Input Layer

$$\begin{bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{bmatrix} + \begin{bmatrix} -0.001 & 0.002717 \\ 0.0019 & -0.00475 \end{bmatrix} = \begin{bmatrix} 0.0989 & 0.402717064 \\ -0.1981 & 0.195245138 \end{bmatrix}$$



$[W]^1$ = Original wts + required/estimated change in wts [from hidden Layer to output layer].

$$[W]^1 = \begin{bmatrix} -0.03 \\ -0.027 \end{bmatrix} + [W]^0 \begin{bmatrix} 0.2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0.1704 \\ -0.5274 \end{bmatrix}$$

Lets try classification

Classification

```
In [1]: # Jesus is my Saviour!

In [2]: import os

In [3]: os.chdir('C:\\Users\\Dr Vinod\\Desktop\\WD_python')

In [4]: # our exported file will appear here

In [5]: import pandas as pd

In [6]: import numpy as np

In [7]: import matplotlib.pyplot as plt

In [8]: from sklearn.preprocessing import normalize
```

Import data

```
In [9]: data = pd.read_csv("C:/Users/Dr Vinod/Desktop/DataSets1/BreastCancer.csv")

In [10]: data = pd.DataFrame(data)

In [11]: data = data.dropna() # important step

In [12]: data.head(3)
Out[12]:
   Unnamed: 0       Id  Cl.thickness  ...  Normal.nucleoli  Mitoses   Class
0           1  1000025             5  ...                1        1  benign
1           2  1002945             5  ...                2        1  benign
2           3  1015425             3  ...                1        1  benign

[3 rows x 12 columns]

In [13]: data.shape
Out[13]: (683, 12)
```

See data

```
In [14]: data.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 683 entries, 0 to 698
Data columns (total 12 columns):
Unnamed: 0          683 non-null int64
Id                  683 non-null int64
Cl.thickness        683 non-null int64
Cell.size           683 non-null int64
Cell.shape          683 non-null int64
Marg.adhesion       683 non-null int64
Epith.c.size        683 non-null int64
Bare.nuclei         683 non-null float64
Bl.cromatin         683 non-null int64
Normal.nucleoli     683 non-null int64
Mitoses             683 non-null int64
Class               683 non-null object
dtypes: float64(1), int64(10), object(1)
memory usage: 69.4+ KB
```

Create Predictors

```
In [15]: X = data.iloc[:, 2:10]

In [16]: X.shape
Out[16]: (683, 8)

In [17]: X.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 683 entries, 0 to 698
Data columns (total 8 columns):
Cl.thickness       683 non-null int64
Cell.size          683 non-null int64
Cell.shape         683 non-null int64
Marg.adhesion      683 non-null int64
Epith.c.size       683 non-null int64
Bare.nuclei        683 non-null float64
Bl.cromatin        683 non-null int64
Normal.nucleoli    683 non-null int64
dtypes: float64(1), int64(7)
memory usage: 48.0 KB
```

Create Response Variable

```
In [18]: y = data.iloc[:, 11]

In [19]: y.shape
Out[19]: (683,)

In [20]: y.dtype # what is 'o', object
Out[20]: dtype('O')
```

Train & Test Data and Scaling

```
In [21]: #_____________train and test data

In [22]: from sklearn.model_selection import train_test_split

In [23]: X_train, X_test, y_train, y_test = train_test_split(X, y)

In [24]: #_____________pre processing of data

In [25]: from sklearn.preprocessing import StandardScaler

In [26]: scaler = StandardScaler()

In [27]: # Fit only to the training data

In [28]: scaler.fit(X_train)
Out[28]: StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
In [29]: # Now apply the transformations to the data:

In [30]: X_train = scaler.transform(X_train)

In [31]: X_train
Out[31]:
array([[ 1.56880671,  0.27189965,  0.58119997, ...,  1.72488735,
         0.22810623,  1.7032463 ],
       [ 0.16578656, -0.72484941, -0.76681536, ..., -0.72450584,
        -0.1890023 , -0.61606781],
       [-0.53572351, -0.72484941, -0.76681536, ..., -0.72450584,
         0.64521476,  1.7032463 ],
       ...,
       [ 1.91956174,  1.93314807,  1.25520764, ..., -0.45235104,
         1.47943182,  1.37191571],
       [-1.23723359, -0.72484941, -0.76681536, ..., -0.72450584,
        -1.02321937, -0.61606781],
       [ 0.16578656, -0.72484941, -0.42981153, ..., -0.72450584,
        -0.1890023 , -0.61606781]])

In [32]: X_train.shape # look at Variable explorer!
Out[32]: (512, 8)
```

```
In [33]: X_test = scaler.transform(X_test)

In [34]: X_test
Out[34]:
array([[-1.23723359, -0.72484941, -0.76681536, ..., -0.72450584,
        -1.02321937, -0.61606781],
       [ 0.5165416 , -0.39259973, -0.0928077 , ..., -0.72450584,
        -1.02321937, -0.61606781],
       [ 0.86729663,  1.60089839, -0.0928077 , ...,  0.36411335,
         1.47943182,  1.7032463 ],
       ...,
       [-0.88647855, -0.72484941, -0.0928077 , ..., -0.72450584,
        -0.60611083, -0.61606781],
       [-0.18496848,  0.93639902,  0.91820381, ...,  0.63626815,
         1.47943182,  1.37191571],
       [-0.18496848,  2.26539776,  0.24419614, ...,  1.72488735,
         2.31364888,  2.36590747]])

In [35]: X_test.shape
Out[35]: (171, 8)
```

```
In [36]: from sklearn.neural_network import MLPClassifier

In [37]: mlp = MLPClassifier(hidden_layer_sizes=(30,30,30))

In [38]: mlp.fit(X_train,y_train)
C:\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
Out[38]:
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(30, 30, 30), learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

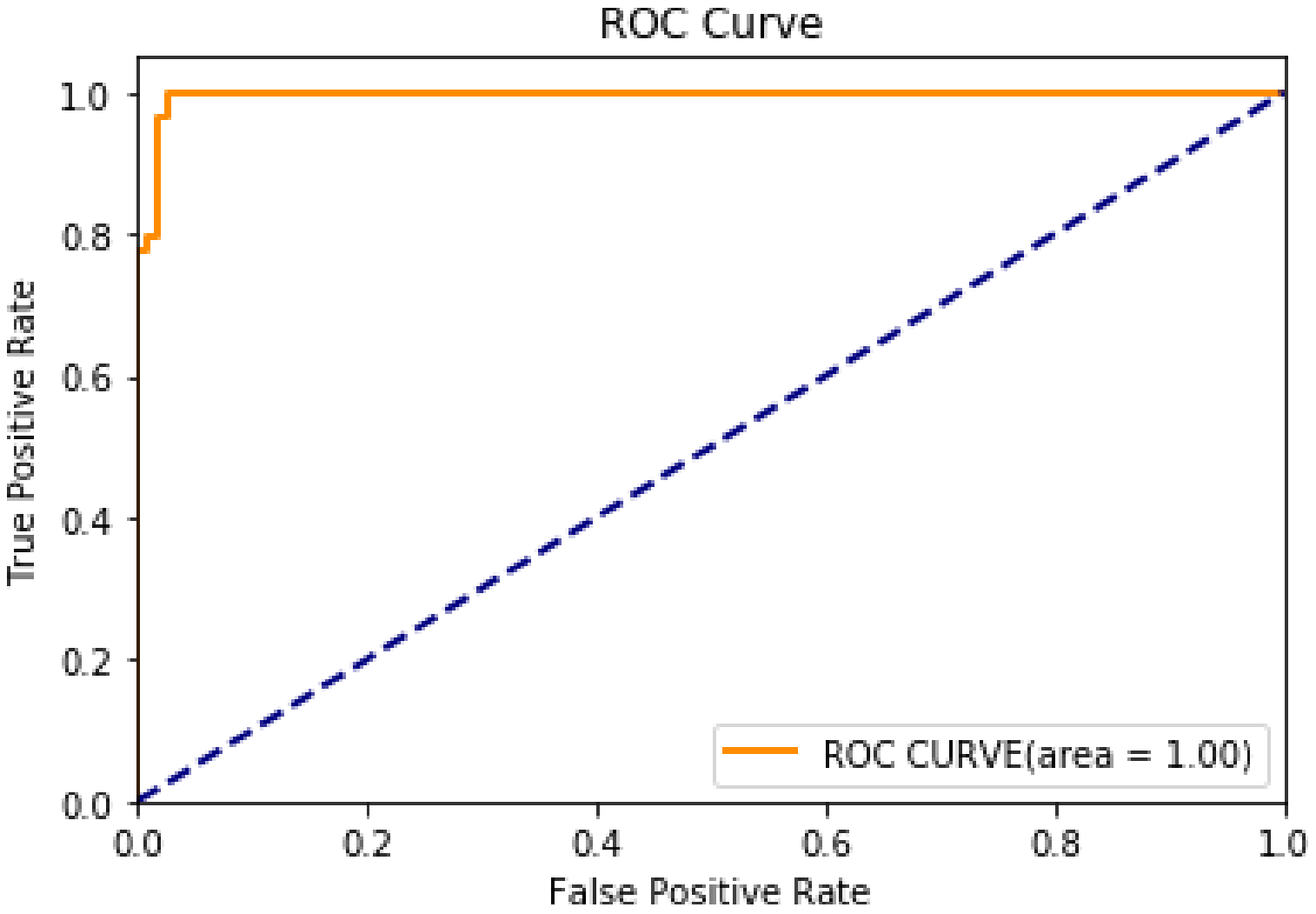
Predict

```
In [39]: #_____________Let's predict our model

In [40]: predictions = mlp.predict(X_test)

In [41]: predictions # too large output!
Out[41]:
array(['benign', 'benign', 'malignant', 'benign', 'benign', 'benign',
       'benign', 'benign', 'benign', 'benign', 'malignant', 'benign',
       'benign', 'benign', 'benign', 'malignant', 'malignant', 'benign',
       'benign', 'benign', 'benign', 'malignant', 'benign', 'malignant',
       'malignant', 'malignant', 'benign', 'benign', 'benign', 'benign',
       'benign', 'benign', 'benign', 'benign', 'benign', 'benign',
       'benign', 'malignant', 'malignant', 'malignant', 'malignant',
       'malignant', 'benign', 'malignant', 'malignant', 'benign',
```

Results

```
In [42]: from sklearn.metrics import classification_report,confusion_matrix

In [43]: print(confusion_matrix(y_test,predictions))
[[115   2]
 [  3  51]]

In [44]: print(classification_report(y_test,predictions))
              precision    recall  f1-score   support

      benign       0.97      0.98      0.98       117
   malignant       0.96      0.94      0.95        54

    accuracy                           0.97       171
   macro avg       0.97      0.96      0.97       171
weighted avg       0.97      0.97      0.97       171
```

```
In [45]: # Curve

In [46]: from sklearn import metrics

In [47]: from sklearn.metrics import roc_curve

In [48]: y_pred_prob = mlp.predict_proba(X_test)[:, 1]

In [49]: fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob, pos_label =
'malignant')

In [50]: auc = metrics.roc_auc_score(y_test, y_pred_prob)

In [51]: plt.figure()
    ...: lw = 2
    ...: plt.plot(fpr, tpr, color = 'darkorange', lw = lw, label = 'ROC CURVE(area
= %0.2f)' % auc)
    ...: plt.plot([0,1], [0,1], color = 'navy', lw = lw, linestyle = '--')
    ...: plt.xlim([0.0, 1.0])
    ...: plt.ylim([0.0, 1.05])
    ...: plt.xlabel('False Positive Rate')
    ...: plt.ylabel('True Positive Rate')
    ...: plt.title('ROC Curve')
    ...: plt.legend(loc = 'lower right')
    ...: plt.show()
    ...: ####
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC CURVE(area = 1.00)

Lets try Prediction

```
In [1]: # Jesus is my Saviour!

In [2]: import os

In [3]: os.chdir('C:\\Users\\Dr Vinod\\Desktop\\WD_python')

In [4]: ## our exported file will appear here

In [5]: import pandas as pd

In [6]: import numpy as np

In [7]: import matplotlib.pyplot as plt

In [8]: import seaborn as sns

In [9]: from sklearn.preprocessing import normalize
```

```
In [10]: #_____________________import data

In [11]: data = pd.read_csv("C:/Users/Dr Vinod/Desktop/DataSets1/insurance.csv")

In [12]: data = pd.DataFrame(data)

In [13]: data = data.dropna() # important step

In [14]: data['sex'].head()
Out[14]:
0    female
1      male
2      male
3      male
4      male
Name: sex, dtype: object

In [15]: data['sex'].dtype # object
Out[15]: dtype('O')
```

```
In [16]: data['sex'].to_frame()
Out[16]:
        sex
0    female
1      male
2      male
3      male
4      male
```

Convert words into numbers

```
In [17]: data['sex']=data.get('sex').replace('male',1)

In [18]: data['sex']=data.get('sex').replace('female',2)

In [19]: c_sex = data['sex'].value_counts()

In [20]: c_sex
Out[20]:
1    676
2    662
Name: sex, dtype: int64
```

Convert words into numbers

```
In [21]: data['smoker']=data.get('smoker').replace('yes',1)

In [22]: data['smoker']=data.get('smoker').replace('no',2)

In [23]: c_smoker = data['smoker'].value_counts()

In [24]: c_smoker
Out[24]:
2    1064
1     274
Name: smoker, dtype: int64
```

```
In [25]: data.head(3)
Out[25]:
   age  sex    bmi  children  smoker     region      charges
0   19    2  27.90         0       1  southwest  16884.9240
1   18    1  33.77         1       2  southeast   1725.5523
2   28    1  33.00         3       2  southeast   4449.4620

In [26]: data.shape
Out[26]: (1338, 7)

In [27]: data.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1338 entries, 0 to 1337
Data columns (total 7 columns):
age         1338 non-null int64
sex         1338 non-null int64
bmi         1338 non-null float64
children    1338 non-null int64
smoker      1338 non-null int64
region      1338 non-null object
charges     1338 non-null float64
dtypes: float64(2), int64(4), object(1)
memory usage: 83.6+ KB
```

```
In [28]: # we dont think that region can be a predictor, so drop this

In [29]: del data['region']

In [30]: data.head(3)
Out[30]:
   age  sex    bmi  children  smoker     charges
0   19    2  27.90         0       1  16884.9240
1   18    1  33.77         1       2   1725.5523
2   28    1  33.00         3       2   4449.4620

In [31]: data.shape
Out[31]: (1338, 6)

In [32]: data.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1338 entries, 0 to 1337
Data columns (total 6 columns):
age         1338 non-null int64
sex         1338 non-null int64
bmi         1338 non-null float64
children    1338 non-null int64
smoker      1338 non-null int64
charges     1338 non-null float64
dtypes: float64(2), int64(4)
memory usage: 73.2 KB
```

```
In [33]: #____________________lets create Predictors and RV

In [34]: X = data.iloc[:, 0:5]

In [35]: X.shape
Out[35]: (1338, 5)

In [36]: X.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1338 entries, 0 to 1337
Data columns (total 5 columns):
age         1338 non-null int64
sex         1338 non-null int64
bmi         1338 non-null float64
children    1338 non-null int64
smoker      1338 non-null int64
dtypes: float64(1), int64(4)
memory usage: 62.7 KB

In [37]: y = data.iloc[:, 5]

In [38]: y.shape
Out[38]: (1338,)

In [39]: y.dtype # float64
Out[39]: dtype('float64')
```

```
In [40]: #_____train and test data

In [41]: from sklearn.model_selection import train_test_split

In [42]: X_train, X_test, y_train, y_test = train_test_split(X, y)

In [43]: #_____pre processing of data

In [44]: from sklearn.preprocessing import StandardScaler

In [45]: scaler = StandardScaler()
```

```
In [48]: # Now apply the transformations to the data:

In [49]: X_train = scaler.transform(X_train)

In [50]: X_train
Out[50]:
array([[-0.09586918, -0.98909239,  1.24993093,  1.61515654, -1.98516667],
       [ 1.18243375, -0.98909239,  1.55984391, -0.90037453,  0.50373604],
       [ 1.5375179 ,  1.0110279 ,  2.16019888, -0.90037453,  0.50373604],
       ...,
       [-0.16688601,  1.0110279 , -0.6955437 , -0.90037453, -1.98516667],
       [ 1.46650107, -0.98909239, -0.80263404, -0.90037453,  0.50373604],
       [-1.37417211,  1.0110279 ,  0.12548229, -0.90037453,  0.50373604]])

In [51]: X_train.shape # look at Variable explorer!
Out[51]: (1003, 5)
```

```
In [52]: X_test = scaler.transform(X_test)

In [53]: X_test
Out[53]:
array([[ 1.18243375e+00,  1.01102790e+00, -3.85630724e-01,
        -9.00374532e-01,  5.03736042e-01],
       [ 3.30231797e-01, -9.89092386e-01,  5.43556550e-04,
         7.76646184e-01,  5.03736042e-01],
       [-1.01908796e+00, -9.89092386e-01, -8.02634043e-01,
        -9.00374532e-01,  5.03736042e-01],
       ...,
       [-2.48523501e-02, -9.89092386e-01, -3.87253305e-01,
        -6.18641740e-02, -1.98516667e+00],
       [ 1.11141692e+00, -9.89092386e-01,  1.14040672e+00,
         1.61515654e+00,  5.03736042e-01],
       [-6.64003815e-01,  1.01102790e+00,  2.78004909e-01,
        -6.18641740e-02,  5.03736042e-01]])

In [54]: X_test.shape
Out[54]: (335, 5)
```

```
In [55]: from sklearn.neural_network import MLPRegressor

In [56]: mlp = MLPRegressor(hidden_layer_sizes=(30,30,30))

In [57]: mlp
Out[57]:
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
             beta_2=0.999, early_stopping=False, epsilon=1e-08,
             hidden_layer_sizes=(30, 30, 30), learning_rate='constant',
             learning_rate_init=0.001, max_iter=200, momentum=0.9,
             n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
             random_state=None, shuffle=True, solver='adam', tol=0.0001,
             validation_fraction=0.1, verbose=False, warm_start=False)
```

```
In [58]: mlp.fit(X_train,y_train)
C:\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
Out[58]:
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
             beta_2=0.999, early_stopping=False, epsilon=1e-08,
             hidden_layer_sizes=(30, 30, 30), learning_rate='constant',
             learning_rate_init=0.001, max_iter=200, momentum=0.9,
             n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
             random_state=None, shuffle=True, solver='adam', tol=0.0001,
             validation_fraction=0.1, verbose=False, warm_start=False)
```

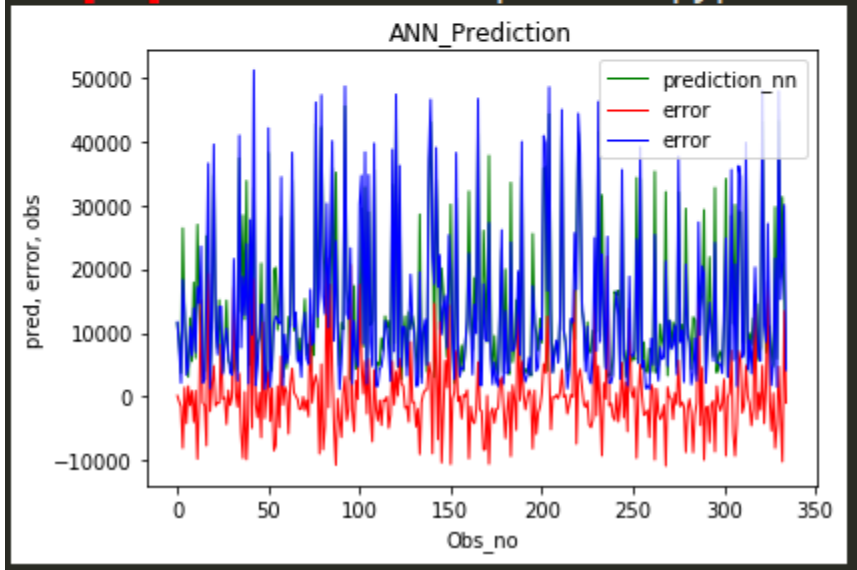
```
In [59]: #_____Let's predict our model

In [60]: predictions = mlp.predict(X_test)

In [61]: error = y_test - predictions

In [62]: error
Out[62]:
837       118.166783
184      -792.811360
192     -1565.733495
1184    -8087.493868
868      1424.136765
46      -4242.032138
```

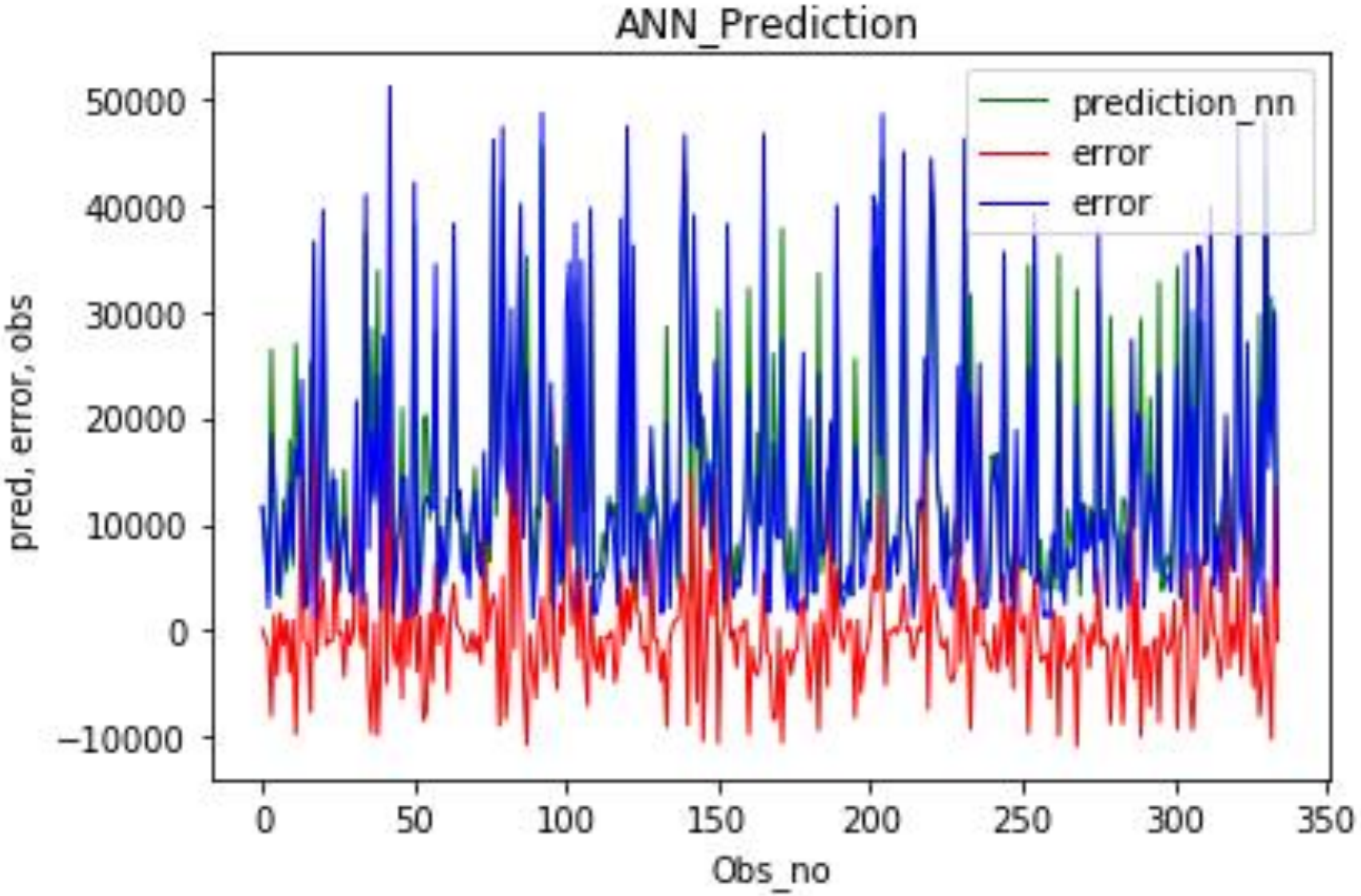
```
In [63]: len(error) # 335
Out[63]: 335
```

```
In [64]: squared_errors = error*error

In [65]: squared_errors
Out[65]:
837     1.396339e+04
184     6.285499e+05
192     2.451521e+06
1184    6.540756e+07
868     2.028166e+06
```

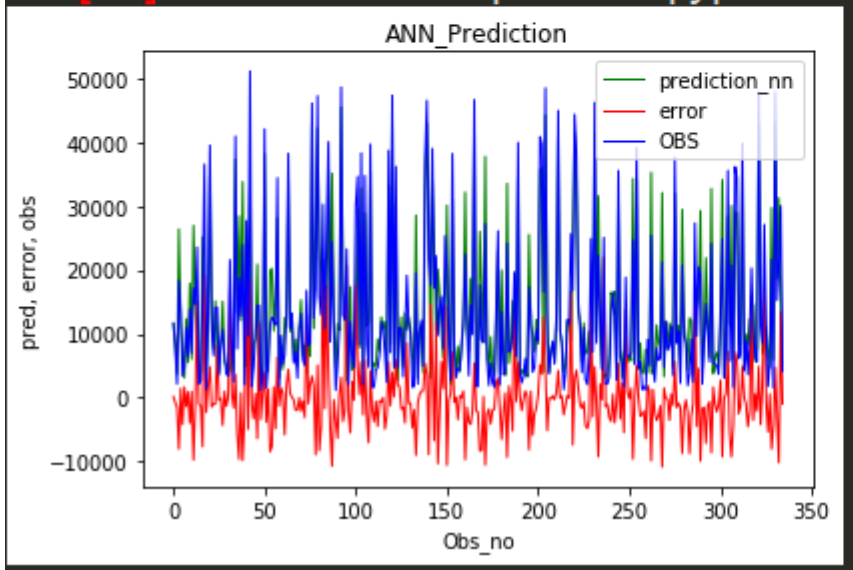
```
In [66]: sum_squared_errors = sum(squared_errors)

In [67]: sum_squared_errors_by_n = sum_squared_errors/335

In [68]: import math

In [69]: RMSE = math.sqrt(sum_squared_errors_by_n)

In [70]: RMSE
Out[70]: 6046.392664728476
```

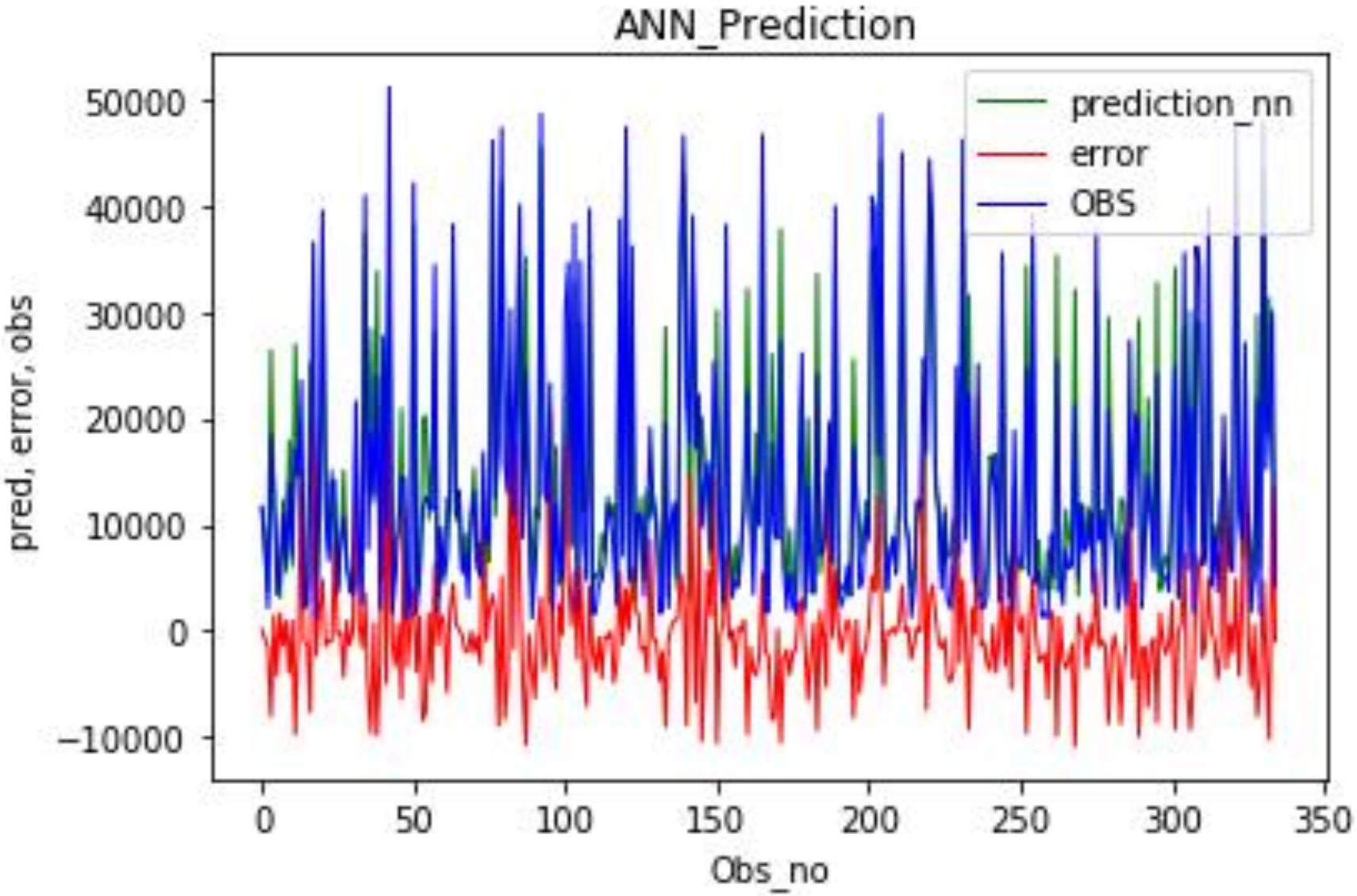
```
In [71]: x = np.arange(0, 335, 1)

In [72]: x
    ...: plt.plot(x, predictions, 'g', label = 'prediction_nn', linewidth = 1)
    ...: plt.plot(x, error, 'r', label = 'error', linewidth = 1)
    ...: plt.plot(x, y_test, 'b', label = 'error', linewidth = 1)
    ...: plt.title('ANN_Prediction')
    ...: plt.xlabel("Obs_no")
    ...: plt.ylabel("pred, error, obs")
    ...: plt.legend()
    ...: plt.show
Out[72]: <function matplotlib.pyplot.show(*args, **kw)>
```




ANN_Prediction
prediction_nn
error
error
pred, error, obs
50000
40000
30000
20000
10000
0
−10000
0
50
100
150
200
250
300
350
Obs_no

```
In [73]: x
    ...: plt.plot(x, predictions, 'g', label = 'prediction_nn', linewidth = 1)
    ...: plt.plot(x, error, 'r', label = 'error', linewidth = 1)
    ...: plt.plot(x, y_test, 'b', label = 'OBS', linewidth = 1)
    ...: plt.title('ANN_Prediction')
    ...: plt.xlabel("Obs_no")
    ...: plt.ylabel("pred, error, obs")
    ...: plt.legend()
    ...: plt.show
Out[73]: <function matplotlib.pyplot.show(*args, **kw)>
```




ANN_Prediction
prediction_nn
error
OBS
pred, error, obs
50000
40000
30000
20000
10000
0
−10000
0
50
100
150
200
250
300
350
Obs_no

Lets compare with Linear Regression

```
In [74]: #_________Linear Regression

In [75]: from sklearn.linear_model import LinearRegression

In [76]: from sklearn import metrics

In [77]: regressor = LinearRegression()

In [78]: lin_reg = regressor.fit(X_train, y_train) #training the algorithm

In [79]: predictions1 = lin_reg.predict(X_test)

In [80]: error1 = y_test - predictions1

In [81]: error1
Out[81]:
837       327.820881
184     -2237.620100
192      -342.472798
```

```
In [82]: len(error1) # 335
Out[82]: 335

In [83]: squared_errors1 = error1*error1

In [84]: squared_errors1
Out[84]:
837     1.074665e+05
184     5.006944e+06
192     1.172876e+05
1184    8.092924e+07
```

```
In [85]: sum_squared_errors1 = sum(squared_errors1)

In [86]: sum_squared_errors_by_n1 = sum_squared_errors1/335

In [87]: import math

In [88]: RMSE1 = math.sqrt(sum_squared_errors_by_n1)

In [89]: RMSE1
Out[89]: 6325.716772924798
```


Don't worry.
Be happy.
Keep learning.